Dominik Reinhardt

Virtualisierung eingebetteter Echtzeitsysteme im Mehrkernbetrieb zur Partitionierung sicherheitsrelevanter Fahrzeugsoftware

Virtualisierung eingebetteter Echtzeitsysteme im Mehrkernbetrieb zur Partitionierung sicherheitsrelevanter Fahrzeugsoftware

Dominik Reinhardt

Universitätsverlag Ilmenau
2016

Impressum

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Diese Arbeit hat der Fakultät Informatik und Automatisierung als Dissertation vorgelegen

Tag der Einreichung:	24. Juni 2015
1. Gutachter:	Prof. Dr. Ing. Winfried Kühnhauser (Technische Universität Ilmenau)
2. Gutachter:	Prof. Dr. Uwe Baumgarten (Technische Universität München)
3. Gutachter:	Prof. Dr. Markus Kucera (Ostbayerische Technische Hochschule Regensburg)
Tag der Verteidigung:	12. April 2016

Technische Universität Ilmenau/Universitätsbibliothek
Universitätsverlag Ilmenau
Postfach 10 05 65
98684 Ilmenau
www.tu-ilmenau.de/universitaetsverlag

Herstellung und Auslieferung
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster
www.mv-verlag.de

ISBN 978-3-86360-135-5 (Druckausgabe)
URN urn:nbn:de:gbv:ilm1-2016000182

Titelphoto: photocase.com

Danksagung

An dieser Stelle möchte ich allen, die zum Gelingen dieser Arbeit beigetragen haben, meinen Dank aussprechen.

Mein besonderer Dank gilt Prof. Dr.-Ing. Winfried Kühnhauser für seine konstruktiven Ratschläge und die Betreuung als Doktorvater meiner Arbeit. Durch ihn wurde diese Arbeit in die richtigen Bahnen gelenkt. Ebenso danke ich Prof. Dr. Uwe Baumgarten für die Übernahme des Zweitgutachtens. Weiterhin möchte ich sowohl Prof. Dr. Markus Kucera für die Übernahme des Drittgutachtens, als auch Prof. Dr. Thomas Waas meinen besonderen Dank aussprechen. Sie haben mich seitens meiner ehemaligen Hochschule bestens unterstützt und standen mir mit ihren Erfahrungen aus Industrie und Forschung stets zur Seite.

Ebenso gilt meine Anerkennung meinen Kollegen Ingo Birner, Matthias Rehm, Dr.-Ing. Matthias Traub, und Andre Lajtkep für ihre wertvollen Diskussionen während meiner Bearbeitungszeit. Gleichzeitig danke ich meinem damaligen Abteilungsleiter Simon Fürst für die Genehmigung und Ermöglichung dieser Arbeit. Auf diesem Weg erwähne ich meine Kollegen des Teams Plattform Software und Systemfunktionen, die für mich den laufenden Kontakt zur Serienentwicklung hergestellt haben.

Da diese Arbeit im Rahmen des ARAMiS Forschungsprojektes entstanden ist, möchte ich mich auf diesem Weg bei allen Partnern für die Zusammenarbeit und die vielen wertvollen Anregungen danken. Besonders bedanke ich mich bei meinen Kollegen Hans-Ulrich Michel und Dirk Kaule aus der BMW Forschung und Technik GmbH für die Beratungsleistung zu meinen Forschungstätigkeiten sowie die hervorragende Zusammenarbeit. Mein Dank gilt sowohl Rakshith Amarnath und Uwe Hartmann von der Robert Bosch GmbH, als auch Dr. Gary Morgan und Dr. Nuria Mata von ETAS Ltd. für die vielen wertvollen Diskussionen zum Thema Virtualisierung. Michael Scheffer und Jens Harnisch von der Infineon Technologies AG danke ich für die vielen Hilfestellungen zur Bereitstellung und Inbetriebnahme der Hardware.

Natürlich möchte ich auch meinen Studenten für die gute Zusammenarbeit und die vielen konstruktiven Vorschläge meine Anerkennung aussprechen. Ich danke somit Andreas Graf, Andre Sekulla, Andre Suhartha, Christoph Müller, Claus Grässle, Damla Olkun, Jennifer Reinelt, Maximilian Güntner und Simon Obermeir.

Insbesondere danke ich Fridolin Röder und Thomas Buchner für die Korrekturleistung zur Finalisierung meiner Arbeit. Nicht zuletzt möchte ich meiner Familie und Heike Benninghoff für das große Verständnis, die Geduld und die moralische Unterstützung meinen tiefsten Dank aussprechen. Ohne deren Mithilfe wäre die Anfertigung dieser Arbeit nicht möglich gewesen!

Kurzfassung

Die Automobilindustrie verzeichnete innerhalb der letzten Jahre einen enormen Zuwachs an neuen elektrischen und elektronischen Fahrzeugfunktionen. Dies führt gleichzeitig zu einer Mehrung der Softwareumfänge in eingebetteten Systemen. Nichtfunktionale Anforderungen wie Sicherheit, Performanz, Verlässlichkeit und Wartbarkeit stellen zusätzliche Herausforderungen an die Entwicklung zukünftiger Fahrzeugsysteme dar. Um die Anzahl der Steuergeräte zu reduzieren, sollen Fahrzeugfunktionen auf gemeinsamen Integrationssteuergeräten konsolidiert werden. Systemvirtualisierung kann hierfür eine zielführende Herangehensweise darstellen, um die Softwaremigration auf Integrationssteuergeräte zu erleichtern und gleichzeitig den geforderten Isolationsansprüchen neuer Sicherheitsstandards gerecht zu werden.

In dieser Arbeit wird die Partitionierung sicherheitsrelevanter Fahrzeugfunktionen auf einer gemeinsamen Hardwareplattform fokussiert. Unter Verwendung von Methoden zur Bewertung sozialer Netzwerke wird eine graphenbasierte Herangehensweise vorgestellt, um die Partitionierbarkeit von Softwarenetzen mit sicherheitsrelevanten Anteilen abschätzen zu können. Zur Realisierung der Systempartitionierung wird eine Methodik zur Auswahl der geeignetsten Kernelarchitektur eingeführt. Dabei werden aus gewählten nicht-funktionalen Eigenschaften potentielle technische Lösungskonzepte innerhalb einer Baumstruktur abgeleitet und ingenieurmäßig bewertet.

Darauf aufbauend wird ein Hypervisor für eingebettete Echtzeitsysteme der Firma ETAS Ltd. evaluiert. Um die Kosten einer zusätzlichen Hypervisorschicht beurteilen zu können, werden in diesem Rahmen Laufzeitmessungen durchgeführt. Somit werden die Auswirkungen einer zusätzlichen Virtualisierungsschicht auf Fahrzeugsoftwaresysteme zur Erfüllung ausgewählter nicht-funktionaler Eigenschaften aufgezeigt.

Die Anbindung virtualisierter Systeme an die Kommunikationsschnittstellen des Hypervisors stellt einen weiteren Schwerpunkt dar. Virtuelle Steuergeräte tauschen sich weiterhin über bereits implementierte Kommunikationskanäle aus und greifen auf gemeinsame Hardwareressourcen zu. Es wird somit ein Konzept eingeführt, um sicherheitsrelevante Anteile des AUTOSAR Microcontroller Abstraction Layers zu entkoppeln. Der Hypervisor selbst wird hierzu an relevanten Stellen erweitert und ein verlässliches Kommunikationskonzept implementiert.

Ein Demonstratoraufbau, zur Konsolidierung von produktiver Fahrzeugsoftware auf einer gemeinsamen Hardwareplattform, finalisiert die Arbeit. Hierfür werden unabhängige Softwarestände paravirtualisiert. Als Resümee der Arbeit erhält der Leser sowohl einen technischen Überblick über den Mehrwert als auch der Kosten paravirtualisierter Fahrzeugplattformen, welche auf Kleinststeuergeräten integriert sind.

Abstract

Within the automotive industry, electric and electronic functionality is rapidly rising within the last few years. This fact yields an increase of software functionality of embedded systems within the car. Non-functional requirements like safety, performance, reliability or maintainability represent additional challenges for future vehicle system development. Vehicle functionality is consolidated on common hardware platforms, to reduce the amount of electronic control units. System virtualization can act as a proper approach, to ease the migration of different vehicle applications to a consolidated system and achieve additional demands for functional isolation.

Within this thesis, the partitioning of safety-related automotive applications on a common hardware platform is focused. To assess the partitioning of safety-related automotive systems, methods for social network evaluation with a graph-oriented approach are proposed. For realizing the system partitioning, a decision-making model is introduced, which results in the most appropriate kernel architecture. From a chosen set of non-functional requirements, technical solutions are derived and rated from a tree structure.

As a result, a hypervisor for embedded real-time systems, supplied by ETAS Ltd., is evaluated. For that purpose, timing measurements are performed to estimate the costs of virtual electronic control units. The impact of an additional virtualization layer for automotive software systems to achieve non-functional requirements is analyzed.

A further main focus is the integration of virtualized systems to the communication interfaces of the hypervisor. Virtual ECUs further exchange information over already implemented communication channels and use common hardware ressources. Thus, a concept to decouple the safety-related parts of the AUTOSAR Microcontroller Abstraction Layer is introduced. The hypervisor itself will be enhanced by a reliable communication concept.

A demonstrator to consolidate already productive automotive applications on a common hardware platform finalizes the work. Here, independent software parts are paravirtualized. This thesis concludes with a technical overview of the benefits and costs for integrating paravirtualized electronic control units on less capable hardware platforms.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Problemstellung . . . 1
- 1.2 Zielsetzung der Arbeit . . . 3
- 1.3 Lösungsweg . . . 4
- 1.4 Gliederung der Arbeit . . . 5

2 Grundlagen und Stand der Technik **7**
- 2.1 Betriebssystemkerne und deren Architekturen . . . 7
- 2.2 Grundlagen der Virtualisierung . . . 8
  - 2.2.1 Virtualisierte Systeme und deren Anforderungen . . . 8
  - 2.2.2 Hypervisortypen . . . 9
- 2.3 Automotive Open System Architecture (AUTOSAR) . . . 10
- 2.4 Funktionale Sicherheit für Straßenfahrzeuge . . . 11
- 2.5 Der Infineon TriCore AURIX Mikrocontroller . . . 12

3 Analyse zur Partitionierung sicherheitsrelevanter E/E-Funktionscluster **15**
- 3.1 Herausforderung und Problemstellung . . . 15
- 3.2 Zentralisierte Bordnetzarchitektur . . . 16
- 3.3 Domänen-orientierte Bordnetzarchitektur . . . 17
  - 3.3.1 Segmentierung von Funktionen und Bildung von Funktionsclustern auf Architekturebene . . . 18
  - 3.3.2 Forderungen der ISO26262 an zukünftige E/E-Systeme . . . 19
  - 3.3.3 Softwareintegration mit gemischter Integritätseinstufung . . . 20
- 3.4 Verlagerung von E/E-Funktionen . . . 22
  - 3.4.1 Skalierbarkeitsanforderungen an zukünftige E/E-Systeme . . . 22
  - 3.4.2 Identifizierte Integrationsszenarien . . . 23
- 3.5 Zusammenfassung des Kapitels . . . 25

4 Bewertung der Partitionierbarkeit sicherheitsrelevanter Softwarekomponenten **27**
- 4.1 Herausforderung und Problemstellung . . . 27
- 4.2 Verwandte Methoden und Techniken . . . 28
- 4.3 Der PageRank . . . 29
- 4.4 Der TrustRank . . . 30

4.5 Der Software Component Rank . . . 31
4.5.1 Bewertung von Fahrzeugsoftwarekomponenten . . . 32
4.5.2 Komplexität und Konvergenz . . . 34
4.5.3 Berücksichtigung sicherheitsrelevanter Softwarekomponenten . . 34
4.5.4 Gewichtete Beziehungen zwischen Softwarekomponenten . . . 36
4.6 Analyse von ECU internen Softwareabhängigkeiten . . . 37
4.7 Bewertung und Einsatzgebiete . . . 39
4.8 Zusammenfassung des Kapitels . . . 40

5 Ableitung von Kernelarchitekturen aus nicht-funktionalen Eigenschaften 43
5.1 Herausforderung und Problemstellung . . . 43
5.2 Verwandte Methoden und Techniken . . . 44
5.3 Attribute der Systemzuverlässigkeit . . . 46
5.3.1 Verlässlichkeit . . . 46
5.3.2 Verfügbarkeit . . . 47
5.3.3 Wartbarkeit . . . 47
5.3.4 Funktionale Sicherheit . . . 48
5.4 Weitere nicht-funktionale Eigenschaften zur Systembewertung . . . 49
5.4.1 Effizienz . . . 49
5.4.2 Skalierbarkeit . . . 50
5.5 Methode zur Bestimmung geeigneter Kernelarchitekturen . . . 51
5.5.1 Grundstruktur eines NFE-Baumes . . . 51
5.5.2 Graphenbasierte Darstellung und Gewichtung . . . 54
5.5.3 Einführung von Querbeziehungen zur Reduktion der Komplexität 56
5.5.4 Gegensätzliche Zielstellungen nicht-funktionaler Eigenschaften . 57
5.6 Ableitung Technischer Maßnahmen und Kernelarchitekturen am Beispiel funktionaler Sicherheit . . . 59
5.6.1 Fehlerpotentiale . . . 60
5.6.2 Technische Maßnahmen . . . 60
5.6.3 Auswertung der gewichteten Ergebnisse . . . 65
5.7 Zusammenfassung des Kapitels . . . 70

6 Kostenanalyse zur Softwareintegration sicherheitsrelevanter Fahrzeugfunktionen 73
6.1 Herausforderung und Problemstellung . . . 73
6.2 Verwandte Methoden und Techniken . . . 75
6.3 Monolithische Systemintegration . . . 76
6.3.1 Portabilität und Wartbarkeit . . . 77
6.3.2 Funktionale Sicherheit . . . 78
6.3.3 Skalierbarkeit . . . 79
6.4 Einsatz eines Mikrokerns . . . 81
6.4.1 Portabilität und Wartbarkeit . . . 81
6.4.2 Funktionale Sicherheit . . . 82
6.4.3 Skalierbarkeit . . . 83

6.5 Systemvirtualisierung . . . 83
6.5.1 Portabilität und Wartbarkeit . . . 83
6.5.2 Funktionale Sicherheit . . . 86
6.5.3 Skalierbarkeit . . . 87
6.6 Analyse einer exemplarischen Hypervisorlösung . . . 88
6.6.1 Richtlinien zur Migration virtualisierter Plattformen . . . 88
6.6.2 Grundlegender Aufbau des Hypervisors . . . 89
6.6.3 Ausnahmebehandlungen von Traps und Interrupts . . . 91
6.6.4 Systemzustände und Sicherheitsmodell des Hypervisors . . . 93
6.6.5 Systemperformanz . . . 95
6.6.6 Portabilität und Wartbarkeit . . . 102
6.6.7 Funktionale Sicherheit . . . 104
6.6.8 Skalierbarkeit . . . 107
6.7 Systembewertung . . . 111
6.7.1 Effizienz . . . 111
6.7.2 Portabilität . . . 113
6.7.3 Wartbarkeit . . . 113
6.7.4 Funktionale Sicherheit . . . 114
6.7.5 Skalierbarkeit . . . 115
6.8 Zusammenfassung des Kapitels . . . 116

7 Kommunikationskanäle für virtuelle Steuergeräte 119
7.1 Herausforderung und Problemstellung . . . 119
7.2 Verwandte Methoden und Techniken . . . 120
7.3 Anbindung paravirtualisierter Hardwaretreiber . . . 121
7.3.1 Realisierung des Zugriffsschutzes von Hardwareressourcen . . . 122
7.3.2 Integrationsansätze von AUTOSAR Treibermodulen . . . 124
7.3.3 Beurteilung dargestellter Integrationsansätze . . . 128
7.4 Informationsaustausch zwischen virtuellen Maschinen . . . 129
7.4.1 Realisierung eines generischen Kommunikationskanals . . . 129
7.4.2 Schnittstellenbeschreibung zur domänenübergreifenden Interaktion . . . 132
7.4.3 Routing von Informationen unter Verwendung einer privilegierten Domäne . . . 132
7.5 Systemanalyse und Beurteilung der Messwerte . . . 134
7.5.1 Antwortzeit in Relation zur Nachrichtengröße . . . 136
7.5.2 Durchsatzmessungen in Relation zur Nachrichtengröße . . . 138
7.6 Zusammenfassung des Kapitels . . . 141

8 Demonstratoraufbau zur Konsolidierung von virtuellen AUTOSAR-Softwaresystemen 143
8.1 Herausforderung und Problemstellung . . . 143
8.2 Methoden zum Informationsaustausch zwischen virtualisierten AUTOSAR Partitionen . . . 144
8.3 Aufbau eines Hochintegrationsdemonstrators . . . 148

8.3.1 Softwarearchitektur und portierte Fahrzeugsoftware 148
8.3.2 Informationsfluss zwischen virtuellen Maschinen 150
8.3.3 Initialisierung der Hardware 151
8.4 Untersuchung des Demonstratoraufbaus 153
8.4.1 Analyse des Rechenbedarfs 153
8.4.2 Funktionsbeanspruchung innerhalb des Hypervisors 154
8.4.3 Beurteilung der Messergebnisse 155
8.5 Zusammenfassung des Kapitels 156

9 Zusammenfassung und Ausblick **159**
9.1 Ergebnisse der Arbeit 159
9.2 Weiterführende Arbeiten 161

A Abkürzungen **163**

B Abbildungen **167**

1 Einleitung

Das Mooresche Gesetz besagt, dass sich die Komplexität bzw. die Anzahl der Transistoren auf elektronischen Bauteilen ca. alle zwei Jahre verdoppelt [110]. Diese Erkenntnis ist bisher zutreffend und wird in diversen Statistiken bestätigt [140]. Der technologische Fortschritt spiegelt sich in immer weiter wachsenden Taktraten der CPUs, ansteigenden Speichergrößen und in einer Mehrung von verbauten Transistoren wieder. Eine Verbesserung der Hardware bedeutete bisher eine Beschleunigung der darauf betriebenen Software (siehe Abbildung 1.1).

Chiphersteller produzieren auf immer kleineren Siliziumflächen und integrieren stetig mehr Transistoren auf einem einzelnen Bauteil. Kleinere Strukturgrößen bei der Chipfertigung bedingen zudem eine Erschwernis der Isolation einzelner Transistoren. Dadurch entsteht ein erhöhtes Auftreten von Leckströmen, welche während der gesamten Prozessorlaufzeit fließen und den Stromverbrauch steigern. Diese Beobachtung stellt die Gesetzmäßigkeit von Moore in Frage, da Ausmaße von Strukturgrößen der Halbleiterelektronik und thermische bzw. energetische Grenzen erreicht werden [143].

Um dieser Tatsache entgegenzuwirken, bietet sich der Einsatz von mehreren parallel arbeitenden Rechenkernen an, welche trotz niedriger Individualleistung gemeinsam die benötigte Gesamtrechenleistung erbringen [144]. Die zuvor vertikale Vermehrung der Rechenkapazitäten, durch eine Erhöhung der CPU-Taktraten, wird nun durch eine horizontale Vervielfachung parallel arbeitender Recheneinheiten erreicht.

Wie in Abbildung 1.1 dargestellt, erreicht die Einführung von Mehrkernsystemen nun auch die Automobilbranche. Dabei steigt die Anzahl der Rechenkerne stetig, wobei sich Frequenzen einzelner CPUs seit einigen Jahren nicht auffallend erhöhen. Es wird ersichtlich, dass gegenüber klassischen Rechnersystemen, Neuerungen auf Hardwareebene mit einem ungefähren zeitlichen Verzug von etwa sieben Jahren für Fahrzeugsysteme erscheinen. Für eingebettete Systeme mit teils sowohl hohen Sicherheits- als auch Echtzeitanforderungen stellen Mehrkernsysteme für die Fahrzeugentwicklung Herausforderungen als auch Chancen dar [50, 60].

1.1 Problemstellung

In Zukunft wird ein fortlaufender **Anstieg der Komplexität von Fahrzeugfunktionen** erwartet [50, 60]. Weiterhin sind in aktuellen Fahrzeugen bereits bis zu 70 Steuergeräte verbaut [121, 128]. Diese Tatsache spiegelt sich im wachsenden Funktionsumfang, also der zunehmenden Anzahl von Applikationen wider (siehe Abbildung 1.1).

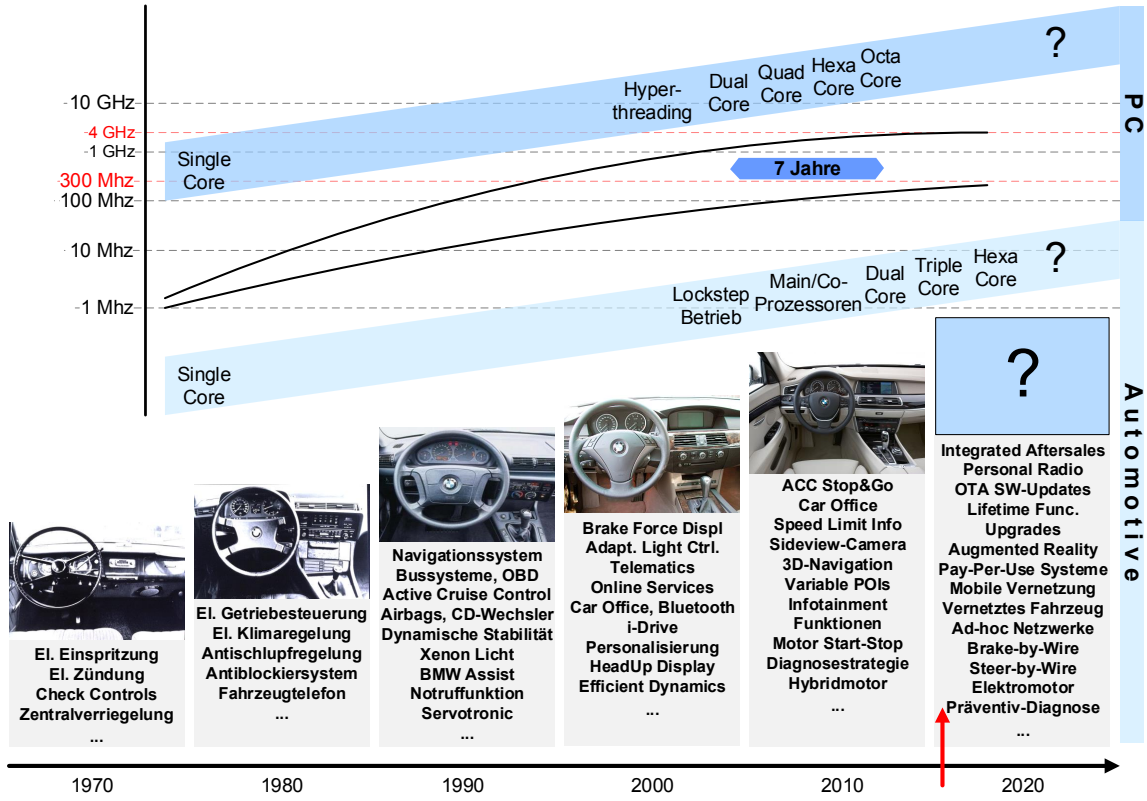


Abbildung 1.1: Vergleich der Rechenleistungen von Steuergeräten der Automobilindustrie mit der Rechenkapazitätssteigerung innerhalb der Consumer-Elektronik am Beispiel des Entwicklungsverlaufs von Kundenfunktionen der BMW AG [58, 128, 141].

Um der wachsenden Anzahl von Steuergeräten entgegenzuwirken und Hardwareressourcen einzusparen, sollen domänenspezifische Fahrzeugfunktionen auf leistungsfähigen Mehrkernsystemen zusammengefasst werden. Dieses Vorgehen wird Hochintegration genannt [69, 121]. Dabei werden mehrere kleinere Steuergeräte auf einer größeren Integrationsplattform zusammengefasst, um Fahrzeugfunktionen darauf zu konsolidieren. Aufgrund des **hohen Vernetzungsgrades zwischen Fahrzeugfunktionen**, können diese nicht problemlos aus bestehenden Softwarearchitekturen herausgelöst und auf unterschiedlichen Steuergeräten integriert werden [107, 123, 135].

Fahrzeugsoftwaresysteme werden aktuell statisch vorkonfiguriert und bedingen einen **erneuten Integrationsaufwand bei Wiederverwendung** bereits bestehender Anwendungen, um geforderte nicht-funktionale Anforderungen (wie Verlässlichkeit, Sicherheit oder Verfügbarkeit) erfüllen zu können. Aufgrund der steigenden Anzahl von Fahrzeugtypen ist die Erstellung von variantenübergreifenden Softwareständen zur Reduzierung von Entwicklungskosten komplex und aufwendig umzusetzen.

In bisherigen Softwaresystemen der Automobilindustrie sind **Anforderungen zum Parallelbetrieb nicht berücksichtigt**. Alle Systeme wurden zuvor sequentiell auf einem Rechenkern ausgeführt. Durch eine vordefinierte Ablauflogik ist die Zugriffsregelung auf Betriebsmittel und Daten somit nicht ausreichend implementiert.

Bereits existierender Quellcode kann daher nicht ohne größere Portierungsaufwände und Schutzkonzepte in den Mehrkernbetrieb überführt werden.

Somit ist die **Partitionierung bereits existierender Software problematisch** und eine optimale Verteilung der Rechenlast innerhalb eines Mehrkernsystems kaum möglich. Fahrzeugfunktionen sind funktional stark miteinander vernetzt und weisen massive Abhängigkeiten zwischen Softwarekomponenten auf [41, 131]. Ungünstig segregierte Mehrkernsysteme verursachen daher eine hohe Kommunikationslast zwischen Rechenkernen und erhöhen dadurch den Rechenaufwand.

Zusätzliche **Entwicklungsrichtlinien der funktionalen Sicherheit** im Rahmen der ISO 26262 [89] schränken die Entwicklung bzw. den späteren Ablauf von sicherheitsrelevanten Fahrzeugfunktionen weiter ein. Als Ziel gilt es, das Gesamtsystem auf ein gewünschtes Sicherheitslevel anzuheben. Dazu muss die Rückwirkungsfreiheit zwischen einzelnen Softwareeinheiten garantiert werden können. Um einer möglichen Fehlerfortpflanzung entgegenzuwirken, werden sowohl zusätzliche Mechanismen zur Fehlerbehandlung als auch ein unabhängiger Funktionsbetrieb gefordert.

Der aktuelle Automobil-Industriestandard für elektrische und elektronische (E/Es) Systeme ist AUTomotive Open System ARchitecture (AUTOSAR) und soll den genannten Problemen durch neue Entwicklungskonzepte und -richtlinien entgegenwirken [29]. Er stellt die zu verwendende, in Schichten aufgeteilte Basissoftware für Fahrzeugsteuergeräte dar und ermöglicht Mehrkerncontroller einzusetzen. Allerdings sind die Vorgehensweisen und Techniken für zukünftige Systeme noch nicht ausreichend verankert [129, 130, 132, 152]. Somit ist die **Erfüllung nicht-funktionaler Anforderungen problematisch**. Nicht-funktionale Anforderungen, wie Sicherheits-, Effizienz- und Adaptionsansprüche sind mit dem derzeitigen Architekturprinzipien des AUTOSAR Standards nicht hinreichend umsetzbar.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist sowohl die Beherrschung der steigenden Softwarekomplexität als auch die fortwährende Erfüllung nicht-funktionaler Anforderungen zur Konsolidierung sicherheitsrelevanter Fahrzeugfunktionen auf einem Integrationssteuergerät.

Hierbei soll, auf Grundlage der Abhängigkeiten innerhalb von Softwarenetzen, zuerst eine **Methode zur Beurteilung der Partitionierbarkeit** bereits existierender Fahrzeugfunktionen erarbeitet werden. Im Vordergrund steht dabei die Hochintegration sicherheitsrelevanter Softwarekomponenten mit unterschiedlichen Sicherheitseinstufungen in einem Mehrkernsystem.

Es soll daraufhin eine **Bewertung unterschiedlicher Architekturansätze** hinsichtlich ihrer Eignung zur Bereitstellung ausgewählter nicht-funktionaler Eigenschaften durchgeführt werden. In diesem Rahmen soll ein neuer Architekturansatz zur Bildung virtueller Steuergeräte innerhalb von Integrationssteuergeräten betrachtet und mit dem vorherrschenden AUTOSAR Standard verglichen werden.

Dabei soll beurteilt werden, ob die **Einführung einer zusätzlichen Virtualisierungsschicht** die Partitionierung bzw. Isolation von Fahrzeugsoftware erleichtert. Die klassischen Konzepte des AUTOSAR-Standards zur Partitionierung von Softwareein-

heiten und deren Kommunikationsmechanismen zwischen den Rechenkernen werden mit den Methoden zur Virtualisierung von Kleinstsystemen verglichen.

Zur weiteren Bewertung paravirtualisierter Fahrzeugsysteme soll eine **Analyse der Virtualisierungskosten** durchgeführt werden. Dabei sollen die Laufzeitkosten unter Verwendung eines eingebetteten, echtzeitfähigen Hypervisor (HV)s ermittelt und anhand ausgewählter nicht-funktionaler Anforderungen dem klassischen Betrieb gegenübergestellt werden. Die resultierenden Zusatzaufwände (wie Integrationsaufwand, Kommunikationskosten oder Ressourcenverbrauch) einer klassischen Systemmigration sind dabei mit der Systemvirtualisierung zu vergleichen.

Zur **Verbesserung des Kommunikationsaustauschs** für virtuelle Maschinen (VMs) sollen auf Grundlage der bereits ermittelten Laufzeitkosten Optimierungsschritte für vorhandene virtualisierte Steuergeräte erarbeitet werden. Dabei gilt als Erwartungshaltung, dass ein nahezu äquivalenter Betrieb für virtualisierte Steuergeräte im Vergleich zu nativen Systemen erreicht wird.

Besonders die **Integration von sicherheitsrelevanten Softwareclustern** mit unterschiedlichen Sicherheitseinstufungen ist bei der Konsolidierung von Fahrzeugsoftware zu berücksichtigen. Dabei soll der rückwirkungsfreie Betrieb, besonders bei gemeinsam genutzten Hard- bzw. Softwarekomponenten, sichergestellt und betreffende sicherheitsrelevante Anteile nicht durch andere Anwendungen gestört werden können.

Zuletzt soll der Nachweis zur **Umsetzbarkeit der erarbeiteten Konzepte** innerhalb eines realitätsnahen Hochintegrationsszenarios erbracht und die Leistungsfähigkeit virtualisierter Fahrzeugsysteme geprüft werden.

1.3 Lösungsweg

Für diese Arbeit werden gemeinschaftlich erarbeitete **Konzepte der domänenorientierten E/E-Architektur** eingeführt [107, 121]. Fahrzeugfunktionen der gleichen Fahrzeugdomäne werden auf logischer Ebene beurteilt und auf technischer Ebene einzelnen Steuergeräten zugeordnet. Domänenleitrechner fungieren hierfür als Integrationssteuergeräte der möglichen Domänen Infotainment, Chassis, Body und Powertrain. In diesem Zusammenhang werden Anforderungen an zukünftige Integrationssteuergeräte erarbeitet und Szenarien zur Konsolidierung von Fahrzeugfunktionen aufgezeigt.

Zur analytischen **Bewertung der Partitionierbarkeit** bestehender Softwarenetze wird der PageRank Algorithmus für die Bedarfe sicherheitsrelevanter Fahrzeugsoftware adaptiert und erweitert [70, 115]. Techniken zur Analyse sozialer Netzwerke eignen sich zur Verarbeitung großer Datenmengen innerhalb eines Graphen, in welchem Knoten (z.B. Softwarekomponenten) in unterschiedlicher Beziehung zueinander stehen [150]. Unter Berücksichtigung sicherheitsrelevanter Fahrzeugfunktionen wird die Vermaschung unterschiedlicher Softwaresysteme in einem Gesamtsystem beurteilt.

Zur Ermittlung des projektspezifisch optimalen Architekturansatzes wird daraufhin durch methodisches Vorgehen ein **Entscheidungsmodell** aufgestellt. Dabei werden nicht-funktionale Eigenschaften durch Baumstrukturen aufgeschlüsselt. Als Er-

gebnis dieses Vorgehens werden sowohl technische Realisierungsvorschläge als auch eine projektspezifische Entscheidung der optimalen Kernelarchitektur (Monolith, Mikrokern, Hypervisor) abgeleitet.

Die zuvor aufgeführten Kernelarchitekturen werden nun anhand nicht-funktionaler Eigenschaften bewertet und gegenübergestellt. Um die Aufwände einer zusätzlichen Virtualisierungsschicht aufzuzeigen, werden **Laufzeitmessungen unter Verwendung eines exemplarischen Hypervisors** für eingebettete Systeme durchgeführt. Als Hardwareplattform wird der Infineon TriCore AURIX Mikrocontroller verwendet. Die gewonnenen Erkenntnisse werden im Anschluss mit den bereits existierenden Konzepten des AUTOSAR Standards abgeglichen.

Für den Informationsaustausch zwischen VMs ist ein virtuelles Kommunikationsmedium notwendig. Daher wird ein **generisches Kommunikationskonzept implementiert und vermessen**. Dieses bildet unterschiedliche Feldbusse Steuergeräte-intern ab und ermöglicht den Austausch von Signalen zwischen VMs.

Um auch die Anforderungen sicherheitsrelevanter Systeme der Automobilindustrie zu erfüllen, werden **Konzepte zur Partitionierung der Treiberschichten** zur Integration virtueller Steuergeräte vorgeschlagen. Hierbei werden bestehende Verfahren der Servervirtualisierung auf eingebettete Fahrzeugsysteme portiert.

Die genannten Lösungsvorschläge werden durch den **Aufbau eines Hochintegrationsdemonstrators** untermauert. Dabei werden ausgewählte Fahrzeugsoftwaresysteme mit unterschiedlicher Integritätseinstufung innerhalb einer virtuellen Umgebung auf dem Infineon Tricore AURIX konsolidiert. Die Machbarkeitsstudie dient der Verdeutlichung, dass - unter Verwendung eines Hypervisors zur Bildung virtueller Steuergeräte - die Hochintegration unterschiedlicher Fahrzeugfunktionen mit unterschiedlichen Sicherheitseinstufungen möglich ist.

1.4 Gliederung der Arbeit

Der weitere Verlauf der Arbeit gliedert sich wie folgt: Kapitel 2 listet grundlegende Begrifflichkeiten und Definitionen auf, welche als Grundlage der Arbeit dienen. In Kapitel 3 wird das grundsätzliche Vorgehen zur Verlagerung und Partitionierung sicherheitsrelevanter Fahrzeugcluster in einer domänen-orientierten Bordnetzarchitektur erläutert. Eine Bewertungsmethode zur Ermittlung und Darstellung konsolidierter Steuergeräte mit sicherheitsrelevanten Softwareanteilen wird in Kapitel 4 eingeführt. Kapitel 5 zeigt ein mögliches Vorgehen zur Identifikation einer geeigneten Kernelarchitektur um nicht-funktionale Eigenschaften konsolidierter E/E-Systeme erreichen zu können. In Kapitel 6 wird der monolithische Kernelansatz des AUTOSAR Standards mit Mikrokern-basierten und Hypervisor-basierten Systemen verglichen. Dabei wird eine Kostenanalyse an einem exemplarischen, echtzeitfähigen HV durchgeführt und auf Basis bestehender Lösungskonzepte gegenübergestellt. Die daraus gewonnenen Erkenntnisse zur Softwarepartitionierung werden in Kapitel 7 vertieft und Kommunikationsmechanismen zwischen virtuellen Maschinen implementiert. Kapitel 8 beschreibt den Demonstratoraufbau und die Integration virtueller AUTOSAR-Steuergeräte. Ka-

pitel 9 fasst die Methoden und Ergebnisse zusammen und gibt einen Ausblick für weiterführende Forschungstätigkeiten.

2 Grundlagen und Stand der Technik

Ziel dieses Kapitels ist die Grundlagen und den Stand der Technik für diese Arbeit zu schaffen. Hierbei soll sowohl auf relevante Kernelarchitekturen, als auch auf virtuelle Systeme eingegangen werden. Weiterhin werden vorherrschende Sicherheits- und Softwarestandards der Automobilbranche aufgeführt, welche für diese Arbeit relevant sind. Die Darstellung der verwendeten Hardware, welche für Evaluierungszwecke und zum Aufbau des Hochintegrationsdemonstrators dient, finalisiert dieses Kapitel.

2.1 Betriebssystemkerne und deren Architekturen

Innerhalb dieser Arbeit soll grundsätzlich zwischen monolithischen, Mikrokern-basierten und Hypervisor-basierten Systemen unterschieden werden [145]. Jede Kernelarchitektur unterscheidet sich unter anderem in Aufbau und Anordnung ihrer Softwareschichten (siehe Abbildung 2.1). Eine der wichtigsten Unterscheidungen der Systeme ist der Umgang mit Berechtigungsstufen zum Zugriff auf die Hardware. Ein Monolith integriert alle Hardwaretreiber die vollständig im Kernel-Modus betrieben werden. Alle Schichten des Betriebssystems bauen dabei aufeinander auf.

Ein Mikrokern (*engl. microkernel*) wird alleinig im höchstprivilegierten Modus (Kernel) betrieben. Treiberschichten greifen durch Schnittstellen auf den Kern zu und erhalten indirekten Zugang auf die Hardware. Ein Mikrokern ist somit ein Teil des Betriebssystems und kann lediglich gleichartige Applikationen auf einem gemeinsamen System betreiben (siehe Abbildung 2.1). Aufgrund der minimalistischen Codegröße wird ein Mikrokern häufig innerhalb von Systemen eingesetzt, welche einen nachweisbar sicheren Betrieb erfordern. Somit wird bei Verwendung eines Mikrokerns grundsätzlich das Ziel verfolgt, die Größe der Trusted Computing Base (TCB) so gering wie möglich zu halten [74, 88]. In der Literatur werden minimalistische Mikrokerne zusätzlich als Nano-, Pico- oder Exokern gekennzeichnet [145].

Ein HV hingegen besitzt starke Ähnlichkeiten zu einem Mikrokern und der Übergang der beiden Architekturen ist je nach Einsatzzweck fließend [74, 76]. Somit wird ein HV identisch zu einem Mikrokern im höchstpriviligierten Modus betrieben. Dabei ist ein HV jedoch kein Bestandteil eines Betriebssystems, sondern ist in einer tieferliegenden Softwareschicht angebracht, um mehrere abgeschlossene Systeme innerhalb von VMs zu betreiben (siehe Abbildung 2.2). Ein HV weist zudem meist eine größere TCB im Vergleich zum Mikrokern auf, und abstrahiert die Hardware von jeweiligen virtualisierten Systemen. Da die Kommunikation zwischen VMs vollständig vom HV kontrolliert wird, ist der Informationsaustausch zwischen einzelnen Partitionen lang-

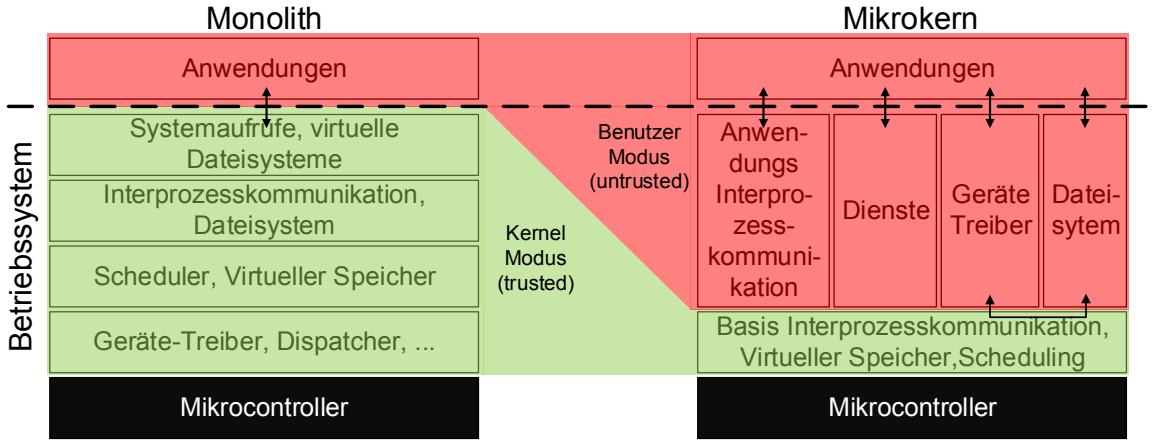


Abbildung 2.1: Kernelaufbau eines mikrokern-basierten bzw. monolithischen Betriebssystems [145].

samer als unter Verwendung eines Mikrokerns. Allerdings erlaubt ein HV die Integration und Betrieb unterschiedlicher Betriebssysteme und schottet diese vollständig voneinander ab. Grundsätzlich wird bei Verwendung eines HV das Ziel verfolgt, den Virtualisierungsaufwand so gering wie möglich zu gestalten [88, 137].

2.2 Grundlagen der Virtualisierung

Im Rahmen dieser Arbeit werden speziell virtualisierte, eingebettete Systeme untersucht. Im Folgenden werden die Grundlagen für dieses Vorhaben aufgezeigt. Erste Experimente im Bereich virtualisierter Systeme wurden bereits von IBM [83] durchgeführt. Goldberg [62] beschreibt Architekturprinzipien und Vorgehen für virtualisierte Systeme und deren Anforderungen. Popek and Goldberg [117] beschreiben initial die Architekturanforderungen an ein System und dessen Instruction Set Architecture (ISA), um darauf einen HV betreiben zu können. Weiterhin wird der grundlegende Aufbau eines HVs definiert.

2.2.1 Virtualisierte Systeme und deren Anforderungen

Popek and Goldberg [117] legen in ihrer Arbeit den Grundstein zur Virtualisierung von Rechensystemen. Vorausgesetzt wird, dass ein virtualisiertes System mindestens zwei Berechtigungsstufen, wie Zugriffsrechte für User bzw. Kernel-Modus, bereitstellen muss. Die ISA und deren Befehle werden dabei in die Gruppen der privilegierten, kontroll-sensitiven, verhaltens-sensitiven und nicht-privilegierten Instruktionen unterteilt. Privilegierte Instruktionen dürfen ausschließlich im Kernel-Modus ausgeführt werden und lösen andernfalls einen System-Trap aus. Werden sensitive Instruktionen im User-Modus ausgeführt, sollen diese ebenfalls einen Trap auslösen. Voraussetzung hierfür ist, dass die Menge aller sensitiven Instruktionen eine Untermenge der privilegierten Instruktionen bildet. Popek and Goldberg [117] beschreiben die folgenden

Anforderungen einer ISA, um eine Prozessorvirtualisierung unter Verwendung eines HV durchführen zu können:

- **Äquivalenz/Genauigkeit:** Das Ausführungsverhalten (zeitlich und funktional) einer Software unter Kontrolle des HVs ist identisch zur nativen Ausführung ohne Virtualisierung.
- **Ressourcenkontrolle/Sicherheit:** Der HV besitzt vollständige Kontrolle aller virtualisierten Ressourcen.
- **Effizienz/Performanz:** Der Hauptanteil zur Ausführung von Maschinenbefehlen erfolgt ohne Eingriff des HVs.

Sind die genannten Anforderungen erfüllt, ist ein System virtualisierbar (*Vollvirtualisierung*). Sind diese Voraussetzungen nicht gegeben, stellen Chiphersteller meist Hardwareunterstützung wie Intel VT bzw. AMD-V zur Verfügung [86]. Kann keine Hardwareunterstützung bereitgestellt werden, müssen oben genannte Voraussetzungen softwareseitig integriert werden. Hierzu muss die zu virtualisierende Software und deren Quellcode an die API des HVs (Virtual Machine Interface (VMI)) angepasst werden. Dieses Vorgehen wird als *Paravirtualisierung* bezeichnet, wobei hauptsächlich das Betriebssystem an die virtualisierte Umgebung angepasst wird, in welcher ausschließlich der HV als höchst privilegierte Instanz im Kernel-Modus betrieben wird. Innerhalb paravirtualisierter Gastsysteme sind alle privilegierten und sensitiven Instruktionen mit Hypercalls zu versehen, die einen Aufruf in den Ausführungskontext des HVs ermöglichen. Im Gegenzug zur Emulation einer ISA ist dies eine sehr effiziente Methode, welche bereits von diversen HVs wie XEN, XtratuM oder Sysgo's PikeOS verwendet wird [34, 93, 103].

2.2.2 Hypervisortypen

Ein HV (auch als *Virtual Machine Monitor* bezeichnet) wird von Goldberg [62] grundsätzlich in die Varianten Typ-1 und Typ-2 HV unterteilt. Diese Unterscheidung wird in Abbildung 2.2 dargestellt [137].

Ein Typ-1 HV ist direkt auf der Hardware integriert (*engl. native, bare metal*) und benötigt kein vollwertiges Betriebssystem. VMs und deren Ressourcenzuteilung werden dynamisch von einem minimalen Betriebssystem unterstützt bzw. von einer Kontrollinstanz verwaltet. Bekannte Vertreter sind u.a. Microsoft Hyper-V oder der Citrix XenServer. Für eingebettete Systeme sind Sysgos PikeOS [76, 93], Green Hills Integrity, Freescales Topaz und der Wind River Hypervisor etabliert. XtratuM und OKL4 von OKLabs sind als quelloffene Software verfügbar [103, 104].

Ein Typ-2 HV wird innerhalb eines vollwertigen Hauptbetriebssystems integriert (*engl. hosted*), welches die letztliche Zuteilung der Ressourcen (CPU, Peripherals, etc.) übernimmt, die vom HV angefragt werden. Der HV kontrolliert von dort ausgehend weitere Gastsysteme. Bekannte Vertreter sind VMware Workstation, VirtualBox oder Linux Kernel-based Virtual Machine (KVM).

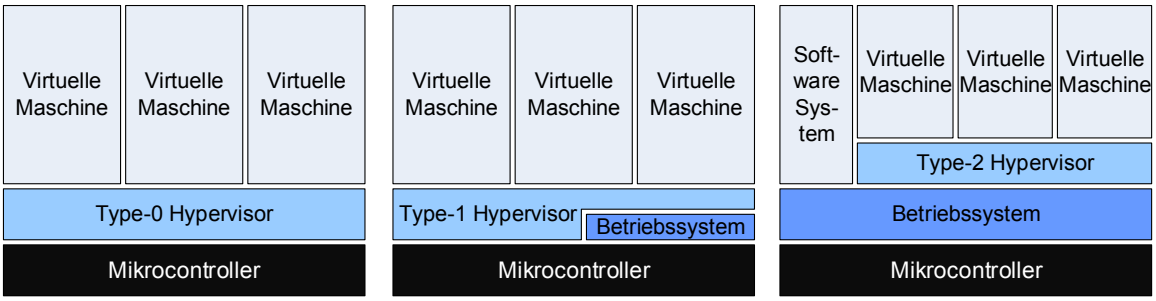


Abbildung 2.2: Klassifizierung möglicher Hypervisorarchitekturen und -typen [145].

Keegan [95] beschreibt einen minimalistischen HV, welcher ohne unterstützendes Betriebssystem arbeitet. Typ-0 HVs werden in der Literatur bisher nicht nennenswert erwähnt und sind nicht klar definiert. Die schlanke Architektur eignet sich besonders für eingebettete Systeme, da ein Typ-0 Hypervisor ohne unterstützendes Betriebssystem integriert wird. RTA-HV von ETAS Ltd. orientiert sich an diesem Konzept [122].

2.3 Automotive Open System Architecture (AUTOSAR)

AUTomotive Open System ARchitecture gilt als aktueller Softwarestandard zur Integration von E/E-Fahrzeugsystemen [30]. Ziel ist die Austauschbarkeit der Software auf unterschiedlichen Steuergeräten durch standardisierte Prozesse und Strukturen zu erhöhen. Zur Weiterentwicklung des Standards wurde eine Entwicklungspartnerschaft aus mehreren Teilnehmern gegründet. Dabei sollen sich Partner alleinig durch die Umsetzung des Standards unterscheiden (Motto: *cooperate on standards - compete on implementation*) [29].

Der AUTOSAR Standard beschreibt die Implementierung der AUTOSAR Basic Software (BSW), die Anbindung von Applikationen und deren Interaktion zwischen einzelnen Modulen. Dabei werden definierte Schnittstellen zwischen Software Components (SWCs) und der Run Time Environment (RTE) mittels einer XML-Beschreibungssprache (ARXML) konfiguriert und mittels Code-Generatoren wird fertiger Quellcode erstellt. Jegliche Kommunikation zwischen Anwendungen wird auf logischer Ebene durch den Virtual Function Bus (VFB) dargestellt und auf technischer Ebene durch das RTE eine Verbindung zwischen Softwareeinheiten umgesetzt.

Die AUTOSAR Architektur ist in diverse Schichten unterteilt, welche in Abbildung 2.3 dargestellt sind. Die gezeigten Schichten abstrahieren die Anwendungsebene von der tatsächlichen Hardware. Soll das Grundsystem auf eine weitere Hardwareplattform migriert werden, muss aufgrund der genannten Schnittstellen lediglich die Microcontroller Abstraction Layer (MCAL) Schicht angepasst werden. *ComplexDriver* ermöglichen dabei eine Integration von Softwareapplikationen, welche nicht auf den AUTOSAR Standard migriert sind. In den meisten Fällen wird die AUTOSAR BSW,

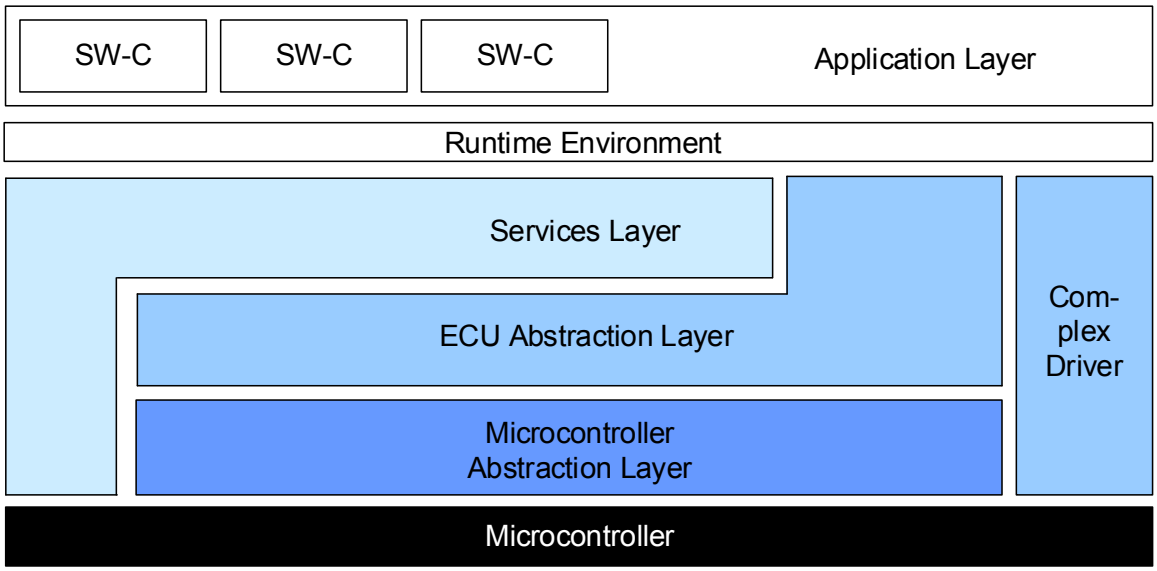


Abbildung 2.3: Softwareschichten der AUTOSAR Systemarchitektur [15].

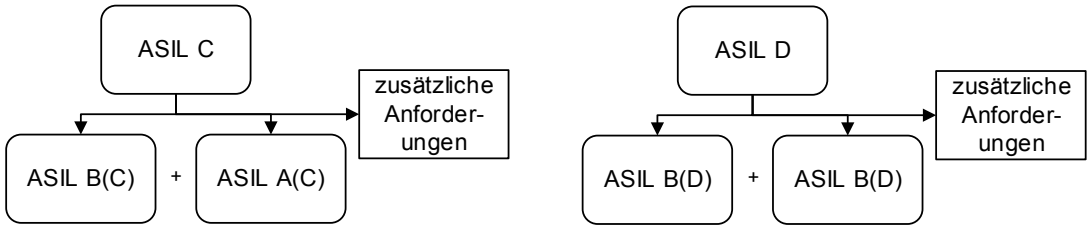
inklusive deren Operating System (OS), also auch die Treiber der MCAL-Schicht, im Kernel-Modus betrieben. Integrierte SWCs können dagegen weiterhin mit Kernel- oder User-Rechten konfiguriert werden (siehe Abbildung 4.1).

2.4 Funktionale Sicherheit für Straßenfahrzeuge

Die ISO 26262 [89] ist der international gültige Standard der Automobilindustrie für Systeme mit erhöhtem Sicherheitsbedarf und stellt einen vollständigen Lebenszyklus eines sicherheitsrelevanten Fahrzeugprojektes dar. Die Norm ist von der IEC 61508 [84] abgeleitet und in zehn Bände unterteilt. Jeder Band ist einem speziellen Teilbereich eines Fahrzeugprojektes unterstellt (wie Hard- oder Software) und stellt Anforderungen an die Entwicklung, an die verwendeten Komponenten und Prozessgestaltung zur Verfügung. Die Zusammenhänge dieser Unterteilungen sind in Abbildung B.3 dargestellt.

Im Prozess der Gefahren- und Risikoanalyse zur Qualifizierung von Fahrzeugfunktionen werden Automotive Safety Integrity Level (ASIL) eingeführt (siehe Band 9, ISO 26262 [89]). Die Einstufungen von A, B, C und D werden vergeben. Dabei stellt ASIL D die höchste Sicherheitseinstufung dar.

Neben den genannten Stufen wird bei nicht-sicherheitsrelevanten Fahrzeugfunktionen lediglich das im Konzern festgelegte Qualitätsmanagement (QM) zur Software- und Systementwicklung betrieben. Innerhalb des technischen Sicherheitskonzeptes wird dargelegt, wie das festgelegte Sicherheitsziel auf technischer Ebene erreicht werden kann. Im Rahmen der funktionalen Sicherheit (*engl. Safety*) betrifft eine Sicherheitseinstufung immer die gesamte E/E-Funktionalität. Dies betrifft sowohl deren Anforderungen und Eigenschaften als auch das Umfeld, in welchem die Fahrzeugfunktion



(a) Dekomposition von ASIL C in ASIL B(C) und ASIL A(C).

(b) Dekomposition von ASIL D in ASIL B(D) und ASIL B(D).

Abbildung 2.4: ASIL Dekomposition laut ISO 26262 [89], zur Aufspaltung hoher Sicherheitseinstufungen in mehrere niedere Einstufungen.

integriert wird. Dieses kann sich unter Umständen auch über mehrere Electronic Control Units (ECUs) hinweg erstrecken.

Die sogenannte ASIL-Dekomposition ist ein Mittel zur Aufspaltung einer Sicherheitseinstufung in mehrere Teilkomponenten mit niederprioreren ASILs. Dieses Vorgehen wird in Abbildung 2.4 dargestellt. Eine sicherheitsrelevante Softwarekomponente eines höheren ASILs kann in mehrere Softwarekomponenten mit niederen ASILs aufgeteilt werden. Dies ermöglicht eine Kostenersparnis oder auch eine potentielle Erleichterung zur Funktionsentwicklung, falls Software- oder Hardwareumgebungen die Integration einer Fahrzeugfunktion mit hohen ASIL nicht zulassen. Dies ist allerdings nur unter Einhaltung vorgegebener Anforderungen möglich. Hierunter fällt der vollständig unabhängige Betrieb der aufgespaltenen Softwareanteile, wodurch diese meist redundant oder gar in diverser Ausprägung integriert sind. Exemplarisch könnte eine Fahrzeugfunktion mit einer Einstufung ASIL C in zwei unabhängige Softwarekomponenten mit niedereren ASILs aufgespalten werden (siehe Abbildung 2.4a).

Innerhalb dieser Arbeit liegt der Fokus auf sicherheitsrelevanten Fahrzeug-Softwaresystemen. Für sicherheitsrelevante Softwaresysteme wird die Rückwirkungsfreiheit (*engl. Freedom from Interference*) zwischen sicherheitsrelevanten und nicht sicherheitsrelevanten Anwendungen gefordert. Die Bedingungen hierfür werden in Part 6 - ANNEX D [89] zur Entwicklung von Software im sicherheitsrelevanten Umfeld aufgeführt. Für einen rückwirkungsfreien Betrieb muss sichergestellt werden, dass sowohl der Ablauf von sicherheitsrelevanten Softwarekomponenten auf zeitlicher, als auch auf örtlicher Ebene nicht gestört wird. Weiterhin muss der korrekte Informationsaustausch zwischen Softwarekomponenten gewährleistet sein. Hierfür werden potentielle Fehlerbilder gelistet, denen durch Absicherungsmaßnahmen vorgebeugt werden soll.

2.5 Der Infineon TriCore AURIX Mikrocontroller

Der Infineon TriCore AURIX Mikrocontroller dient innerhalb dieser Arbeit als Evaluationsplattform für den Demonstratoraufbau und wird in den Kapiteln 6, 7 und 8 verwendet. Der Controller besteht aus insgesamt drei Rechenkernen [4], welche mit einer maximalen Taktfrequenz von 200 MHz betrieben werden können. Jeder Re-

chenkern besitzt lokalen Arbeitsspeicher, der zur Code- und Datenablage verwendet werden kann. Ein globaler Datenspeicher, die Local Memory Unit (LMU), steht jedem Kern gleichermaßen zur Verfügung. Abbildung B.1 zeigt das Blockschaltbild des Systemaufbaus des Infineon AURIX TriCore Mikrocontrollers [4] und enthält detaillierte Informationen über vorhandene Speichergrößen. Jeder Rechenkern fungiert als Bus-Master und ist mit den verbleibenden Rechenkernen über eine Crossbar zum Signalaustausch verbunden.

Die Prozessorarchitektur selbst wurde nicht zur Unterstützung virtualisierte Systeme optimiert [122]. Allerdings werden die Zugriffsrechte zwischen dem Supervisor (SU) (dem Kernel-Modus des AURIX Mikrocontrollers) und dem User-Modus aufgeteilt. Weiterhin unterteilt sich der User-Modus in User-1 und User-0 Berechtigungen. Letztere schränken zusätzlich den Zugriff auf Peripheriecontroller (CAN, FlexRay, etc.) ein [2]. Jeder Prozessor besitzt seine individuell zugewiesenen Register und einen individuellen Adressbereich. Hierbei kontrolliert jeder Rechenkern sein Program Status Word (PSW). Um auf den Zustand bzw. auf das Register des Kerns schreibend zugreifen zu können wird die Instruktion Move To Core Register (MTCR) bzw. lesend die Instruktion Move From Core Register (MFCR) verwendet. MTCR kann lediglich im SU-Modus ausgeführt werden, wobei lesender Zugriff auch im User-Modus gestattet ist [3].

Pro Rechenkern existiert eine exklusive Memory Protection Unit (MPU) zur Umsetzung des Speicherschutzes. Diese unterscheidet zwischen 8 Code- und 16 Datenadressräumen und besitzt unterschiedliche Konfigurationen bzw. Freigaben für diese. Jede MPU verfügt über 4 Konfigurationssets (*engl. range sets*), welche unterschiedliche Zugriffsberechtigungen angeben (lesen, schreiben, ausführen). Ein jeweiliges Konfigurationsset wird innerhalb des Prozessors ausgewählt und schaltet für den aktuellen Zustand gewisse Adressbereiche frei oder schränkt diese entsprechend ein. Eine MPU hat eine Auflösung von 8 Byte. Somit können Adressbereiche, welche innerhalb von 8 Byte direkt aufeinander folgen, nicht durch die MPU voneinander getrennt betrachtet werden. Die Konfiguration der MPU selbst kann ausschließlich im SU-Modus erfolgen [64, 122].

Jeder Prozessor unterstützt Traps (synchrone Ausnahmen) und Interrupts (asynchrone Ausnahmen). Traps werden aufgrund von illegalen Aufrufen und Instruktionen verursacht. Diese beinhalten Verletzungen von Zugriffsrechten, Auslösen von Speicherschutzmechanismen und viele mehr. Interrupts werden beispielsweise durch eingehende Kommunikationssignale ausgelöst. Ein Peripheriecontroller, welcher einen Interrupt auslösen kann, besitzt ein Kontrollregister per Interrupt. Dieser verweist auf den betreffenden Rechenkern, auf welchen die jeweilige Anfrage in der gewünschten Interruptpriorität weitergeleitet werden soll. Jeder Rechenkern definiert anhand seiner eigenen CPU-Priorität, welche Interrupts angenommen oder abgelehnt werden. Pro Kern sind bis zu 255 Interrupts konfigurierbar. Zur Identifikation besitzen Traps und Interrupts eigene Vektortabellen und werden im Speicher durch den Base Interrupt Vector Table Pointer (BIV) bzw. Base Trap Vector Table Pointer (BTV) identifiziert.

Wie in Abbildung B.1 dargestellt, besitzt der AURIX Mikrocontroller diverse Feldbusschnittstellen zur Vernetzung von Steuergeräten. Dies beinhaltet unter anderem

Controller Area Network (CAN), FlexRay und Ethernet. Weiterhin sind diverse Timer und vier Watchdogs (einer Watchdog pro Rechenkern und ein Safety-Watchdog) integriert. Diese Controller sind über den *System Peripheral Bus* Steuergeräte-intern vernetzt.

3 Analyse zur Partitionierung sicherheitsrelevanter E/E-Funktionscluster

Ziel dieses Kapitels ist sowohl die Aufführung bestehender Konzepte einer verteilten als auch einer zentralisierten E/E-Architektur. Diese Konzepte werden mit einer potentiellen Domänen-orientierten E/E-Architektur verglichen, welche der steigenden Anzahl von Steuergeräten entgegenwirken soll. Dabei wird ein analytisches Vorgehen zur Segmentierung von Fahrzeugfunktionen in Funktionscluster vorgestellt, um diese den jeweiligen Steuergeräten korrekt zuordnen zu können. Im Fokus liegt die Partitionierung von sicherheitsrelevanter Fahrzeugsoftware mit unterschiedlichen Integritätseinstufungen. Hieraus ergeben sich neue Anforderungen zur Konsolidierung von Fahrzeugsoftware auf einem gemeinsamen Integrationssteuergerät. Für den weiteren Verlauf der Arbeit werden potentielle Szenarien zur Verlagerung von Fahrzeugsoftware zwischen Integrations- und Substeuergeräten aufgestellt.

3.1 Herausforderung und Problemstellung

Aktuell werden im Rahmen einer verteilten Bordnetzarchitektur mehr als 70 Steuergeräte in Fahrzeugen verbaut (siehe Abbildung B.5) [128]. Diese Tatsache stößt an Grenzen in vielfacher Art und Weise:

Einzelne Steuergeräte integrieren eine Vielzahl von E/E-Funktionen ECU-übergreifend. Steuergeräte werden mit Hilfe von Feldbussen miteinander verbunden [35, 107]. Hierdurch entsteht ein erheblicher logistischer **Integrations- und Vernetzungsaufwand**. Steuergeräte und deren Softwarekomponenten werden sowohl von einem Original Equipment Manufacturer (OEM) selbst als auch von Zulieferern entwickelt.

Aus geometrischer Sicht ist innerhalb eines Fahrzeuges der **Bauraum stark begrenzt**. Steuergeräte können aufgrund des geringen Fahrzeugbauraums nicht in beliebiger Stückzahl verbaut werden. Durch die Vielzahl von Steuergeräten entstehen erhöhte Materialkosten im Bereich des Kabelbaums. Dieser muss arbeitsintensiv für jede Bordnetzvariante individuell angefertigt werden.

Weiterhin entstehen aufgrund der erhöhten Informationsdichte zwischen Steuergeräten **Kommunikationsengpässe** auf vereinzelten Fahrzeugbussen. Dies kann zu zeitlichen Problemen und Verzügen führen. Jedes Steuergerät besitzt gewisse Grundfunktionalitäten, welche wiederkehrend pro ECU integriert sind. Dieser Aufwand ist nur schwer über unterschiedliche Fahrzeugtypen skalierbar, was zusätzliche Kosten

sowohl für Hardware, wie Stromversorgung oder Buscontroller, als auch Software, wie Systemfunktionen und Basissoftware, mit sich bringt.

Die Entwicklung einer **einheitlichen und nachhaltigen Bordnetzarchitektur** stellt für zukünftige E/E-Systeme eine Herausforderung dar. Eine E/E-Architektur muss über einen langen Zeitraum offen gegenüber Änderungen sein und flexibel auf unterschiedlichste Produktausbaustufen angepasst werden können. Innerhalb eines Lebenszyklus einer bestehenden Bordnetzarchitektur müssen aufgrund von Hardwareänderungen, Fahrzeugfunktionen zwischen Steuergeräten mehrmals verlagert werden können.

Die **Skalierbarkeit** der aktuellen, verteilten Bordnetzarchitektur wird über die Hinzunahme bzw. den Wegfall einzelner Hardwarebausteine gesteuert. Wird eine Fahrzeugfunktion nicht benötigt, entfällt im besten Fall ein Steuergerät gänzlich. Besteht der Bedarf an bestimmten Grundfunktionalitäten weiterhin, kann das Steuergerät nicht vollständig eingespart werden.

Unter Verwendung angepasster Hardwarecontroller können unterschiedliche Ausbaustufen (Grund- bzw. Vollausstattung) von E/E-Komponenten realisiert werden. Allerdings bedingt dieses Vorgehen eine gesteigerte **Variantenvielfalt**, was erhöhten Wartungsaufwand mit sich bringt.

Werden zudem **sicherheitsrelevante Fahrzeugfunktionen** auf einem Steuergerät gekapselt, wird die Produktauswahl an potentiellen Hardwarecontrollern durch zusätzliche Sicherheitsanforderungen erschwert.

3.2 Zentralisierte Bordnetzarchitektur

Eine Alternative zur verteilten E/E-Architektur ist die Konsolidierung von Fahrzeugfunktionen auf leistungsfähigere Steuergeräte. Die geringere Anzahl von Steuergeräten reduziert den Vernetzungsaufwand untereinander (kleinere Feldbusse) und spart unter anderem Kosten im Bereich des Kabelbaums ein. Da eine Vielzahl von Softwarekomponenten nun auf einem Steuergerät integriert ist, können Basisfunktionalitäten von mehreren Anwendungen verwendet werden (*engl. Single Source of Information*).

Benachbarte Komponenten innerhalb des Steuergerätes können Timing-Anforderungen leichter standhalten und Informationen direkter über einen gemeinsam genutzten Speicher austauschen. Verhältnismäßig langsame Transportmedien stehen hierbei nicht mehr im Weg. Abhängig von der Leistungsfähigkeit einer ECU können eingehende Signale, wie Bilddaten einer Videokamera, ohne Umwege über viele Rechenknoten verarbeitet werden. Jedoch ist die physische Verbindung zwischen Sensor- und Recheneinheit immer noch notwendig.

Sind viele Softwarekomponenten auf ein Steuergerät zusammengefasst, birgt dies auch Nachteile. So werden leistungsfähigere und kostenintensive Controller benötigt, welche das erhöhte Rechenaufkommen bewältigen können. Die wachsende Komplexität der Systeme fördert die Fehlerausbreitung einzelner Softwareeinheiten, falls diese nicht ausreichend räumlich und zeitlich geschützt werden können. Die Systemskalierbarkeit auf Bordnetzebene verschlechtert sich zusätzlich durch zentralisierte Architekturen.

Ein rationaler Einsatz der Hardware wird erschwert, wenn Funktionalitäten in Steuergeräten in geringeren Ausbaustufen nicht mehr benötigt werden. Es sind Architektur- und Integrationskonzepte notwendig, welche individuelle Fahrzeugmodelle und Ausbaustufen ermöglichen.

Im Fall eines Steuergeräteausfalls, verursacht durch Überhitzung oder Stromausfall, sind neben den direkten Funktionen womöglich viele vernetzte Fahrzeugfunktionen betroffen. Dies zeigt ein grundsätzliches Problem im Bereich der funktionalen Sicherheit auf, da aufgrund von starken Abhängigkeiten zwischen Fahrzeugfunktionen Redundanzkonzepte im Bereich der Hardware erschwert umsetzbar sind.

3.3 Domänen-orientierte Bordnetzarchitektur

Um sich die Vorteile einer verteilten bzw. zentralisierten Bordnetzarchitektur zu Nutze zu machen, wird ein Domänen-orientierter Architekturansatz angedacht [121]. Für zukünftige E/E-Bordnetzarchitekturen müssen somit Fahrzeugfunktionen ihrer Domäne zugeordnet werden. Bisherige E/E-Fahrzeugarchitekturen, welche nicht in Schichten kaskadiert sind, können nur schwer an die ständig wachsende Komplexität angepasst werden.

Im Rahmen einer verteilten Architektur sind Fahrzeugfunktionen fest einem bestimmten Steuergerät zugeordnet. Dies erschwert die Harmonisierung des Gesamtbordnetzes zur Ermöglichung einer einheitlichen Architektur. Grundfunktionalitäten des Fahrzeugs bzw. nicht differenzierende Zulieferkomponenten könnten nur über Umwege eingebunden werden; der Mehrwert ist gering. In einer Domänen-orientierten Architektur sind Fahrzeugfunktionen ihrer spezifischen Domänen untergliedert. Typische Domänen sind Infotainment, Chassis, Powertrain und Driver Assistance [17, 107].

Um die räumliche und zeitliche Skalierbarkeit des Systems zu fördern, werden als zusätzliche Abstraktionsschicht Domänenleitrechner (DLR) (*engl. Domain Controller Unit (DCU)*) oder -controller eingeführt (siehe Abbildung 3.1). Für jede Automobildomäne ist daher ein individueller DLR denkbar. Jeder DLR stellt hierbei ein leistungsfähiges Integrationssteuergerät dar und kontrolliert zugleich angegliederte Feldbusse. Um Energie einzusparen, können verbundene Netzwerk-Cluster individuell deaktiviert werden [134]. Dieses Vorgehen wird als Teilnetzbetrieb bezeichnet [23, 59].

DLRs identifizieren einen bestimmten Fahrzeugzustand und reagieren entweder mit der Aktivierung oder Deaktivierung der betroffenen Substeuergeräte (SUBs) (siehe Abbildung 3.1). DLRs werden mittels schneller Transportmedien miteinander verbunden. Deshalb werden dafür geeignete Netzwerkprotokolle wie Ethernet verwendet. SUBs werden mit ihren jeweiligen DLR mit typischen Feldbussen wie CAN oder FlexRay verbunden.

Weiterhin sollen DLRs als Integrationsplattformen für Softwarekomponenten verwendet werden. Allgemein sind DLRs für OEM spezifische Fahrzeugsoftware bestimmt. Sie agieren unter anderem als Router und sind für die Informationsaustausch der mit ihnen verbundenen SUBs verantwortlich. Im Gegensatz zu den DLRs sind diese Steuergeräte mit weniger Hardwareressourcen ausgestattet. Auf ihnen werden vorwiegend Fahrzeugfunktionalitäten von Zulieferern integriert. In erster Linie sollen

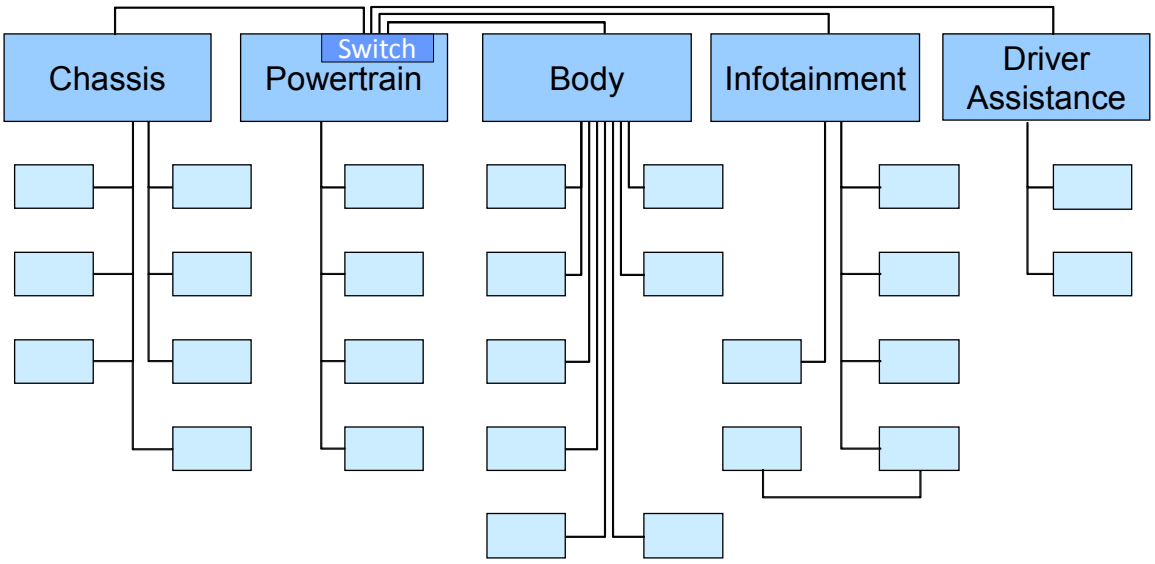


Abbildung 3.1: Darstellung einer möglichen Domänen-orientierten Architekturauslegung von Integrationssteuergeräten bzw. Domänenleitrechnern und einzelnen SUB-Steuergeräten [123].

SUBs Standardkomponenten darstellen und leicht verlagert bzw. ausgetauscht werden können.

Hard- und Softwaresysteme, auf welchen eine Vielzahl von Komponenten zentralisiert sind, werden als hochintegrierte Systeme bezeichnet (Substantiv: *Hochintegration*). Parallel dazu wird der englische Begriff *Large Scale Software Integration (LSSI)* eingeführt, welcher sich vom bereits gebräuchlichen Begriff *Very Large Scale Integration (VLSI)* ableitet [121]. Im Rahmen dieser Arbeit wird angenommen, dass ein hochintegriertes System sicherheitsrelevante Software gemischter ASIL-Einstufung (*engl. mixed integrity*) auf einem Steuergerät vereint. Um eine Behinderung bzw. Beeinflussung zwischen den Anwendungen zu vermeiden, müssen diese voneinander isoliert werden.

3.3.1 Segmentierung von Funktionen und Bildung von Funktionsclustern auf Architekturebene

Die Verlagerung von Fahrzeugfunktionen erfolgt in mehreren Schritten. Wie in Abbildung 3.2 dargestellt, wird die Einigkeit von Fahrzeugsoftware zuerst aus logischer Sicht betrachtet. E/E-Funktionen werden innerhalb des Bordnetzes ihrer individuellen Fahrzeugdomäne zugeordnet. Im Folgenden werden Fahrzeugfunktionen entweder einem DLR oder seiner SUBs zugeordnet. Die dort angeordneten Funktionen, welche bereits stärkere oder auch schwächere Beziehungen zu benachbarten Einheiten aufweisen, formen eine funktionelle Gruppe. Datenflusspfade müssen kurz und komprimiert realisiert sein.

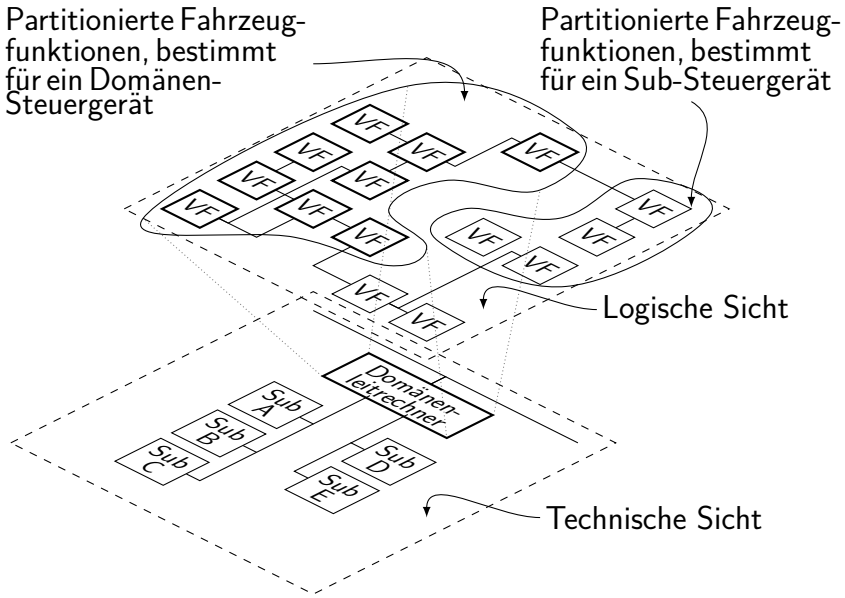


Abbildung 3.2: Logische Segmentierung eines Bordnetzes in funktionale Cluster, zur Verteilung von Vehikelfunktionen auf Domänen- oder Substeuergeräte [121].

Fahrzeugsoftware muss leicht zwischen den Steuergeräten verschiebbar sein und eine lose Koppelung untereinander aufweisen. Der Grad der Zusammengehörigkeit anhand dieser Schnittstellen soll nachvollziehbar gestaltet sein. Hingegen müssen sich Anwendungen von vormals eng zusammenhängenden Fahrzeuganwendungen durch wohldefinierte Schnittstellen klar trennen lassen. Dies wird bereits durch den AUTOSAR Standard vorgegeben [21, 22]. Schnittstellen zwischen Domänen bzw. DLRs werden Domänenschnittstellen genannt [17].

Zur Erstellung einer E/E-Architektur wird entschieden, ob eine Anwendung auf einem Domänenleitrechner oder auf einer Sub-ECU integriert wird. Dabei müssen sich Reibungsverluste zur Softwareintegration für unterschiedliche Fahrzeugausbaustufen, inklusive späterer Innovationssprünge, in Grenzen halten. Fahrzeugsoftware soll also flexibel auf unterschiedlichen Steuergeräten verlagert werden können. Der beschriebene Vorgang ist nochmals in Abbildung B.4 dargestellt. Der iterative Prozess zur Segmentierung der Anwendungsgruppen setzt sich über jede Architekturebene ähnlich fort. Die Ermittlung der Zugehörigkeiten einzelner Fahrzeugfunktionen und deren Softwarekomponenten wird dabei Schritt für Schritt verfeinert.

3.3.2 Forderungen der ISO26262 an zukünftige E/E-Systeme

Eine vereinfachte Verlagerung von Funktionalitäten birgt jedoch zusätzliche Einschränkungen. Neben diversen funktionalen Anforderungen existieren eine Vielzahl nicht-funktionaler Anforderungen. Funktionale Sicherheit bezieht sich auf die Fahrzeugfunktion bzw. Funktionalität und beschränkt sich nicht auf einzelne Systemkomponenten. Trotzdem können einzelne Bausteine im Gesamtsystem qualifiziert werden, um deren korrekte Funktionsweise zu sichern. Eine finale Prüfung des Systemverbun-

des ist jedoch weiterhin notwendig. Die ISO 26262 [89] fordert hierfür grundsätzlich die Rückwirkungsfreiheit zwischen Fahrzeugfunktionen in **Raum**, **Zeit** und während des **Informationsaustauschs**.

Für den weiteren Verlauf der Arbeit werden die im Folgenden aufgeführten Annahmen der ISO 26262 [89] ausgewählt und für zusätzliche Analysen herangezogen [123]. Da sich diese Arbeit auf die Bedarfe von Softwaresystemen konzentriert, werden Anforderungen an die Hardware nicht behandelt.

Ausgewählte Sicherheitsanforderungen zur Herstellung der Rückwirkungsfreiheit:

- Softwarekomponenten müssen lose gekoppelt sein.
- Speicherschutzkonzepte müssen vorhanden sein.
- Adäquates Scheduling für den Betrieb sicherheitsrelevanter Software.
- Monitoring Konzepte zur Ablaufplausibilisierung.
- Unabhängige Ausführungspfade der Ergebnisermittlung.
- Degradationkonzepte im Fehlerfall.

Zusätzliche Anforderungen an Domänenleitrechner:

- Wiederverwendbarkeit muss sichergestellt sein.
- Integrierte Fahrzeugfunktionen müssen weiterhin dekomponierbar sein.
- Funktionscluster und -partitionen müssen neugestartet werden können.
- Effizienter Ablauf von sicherheitsrelevanten Anwendungen.
- Nachträglich integrierte Software muss isolierbar und geschützt sein.
- Rückwärtskompatibilität muss gewährleistet sein.

3.3.3 Softwareintegration mit gemischter Integritätseinstufung

Ein ASIL eines Sicherheitszieles einer sicherheitsrelevanten Fahrzeugfunktion reicht durch den gesamten Entwicklungsprozess der damit verbundenen Anwendungen. Somit ist die jeweilige Sicherheitseinstufung nicht auf eine einzelne Softwarekomponente beschränkt. Verknüpfte Hard- bzw. Softwarekomponenten werden somit in die Sicherheitsbetrachtung mit einbezogen und der vergebene ASIL an diese vererbt. Dies stellt sicher, dass alle relevanten Komponenten unter gegebenen Bedingungen zur korrekten Erfüllung der Gesamtfunktion beitragen.

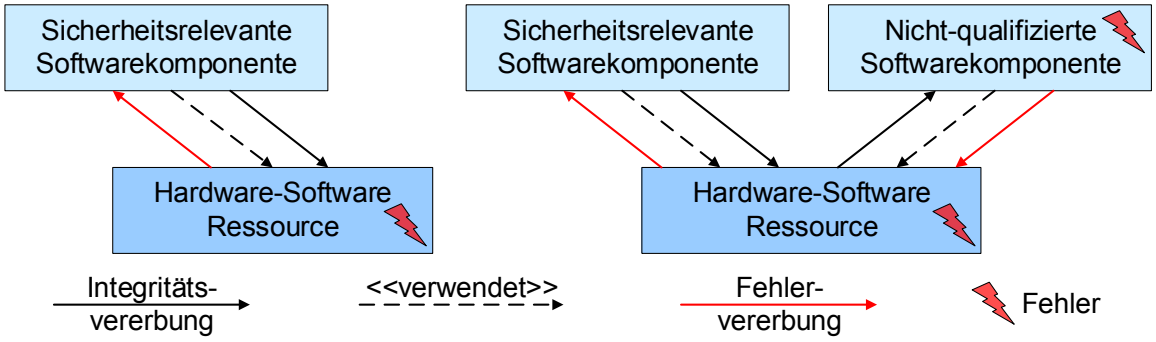


Abbildung 3.3: Fehlerfortpflanzung zwischen qualifizierten und nicht-qualifizierten Komponenten.

Alle Anwendungen, die notwendig sind um eine gewisse Sicherheitseigenschaft zu erfüllen, bilden die TCB eines Systems. Dies betrifft in erster Linie die sicherheitsrelevanten Softwarekomponenten der Fahrzeugfunktion selbst und alle verknüpfte Softwareanteile, Bibliotheken und Dienstfunktionen. Dies umfasst auch das Betriebssystem und mögliche Hardwaretreiber innerhalb des AUTOSAR MCALs (siehe Abbildung 4.1). Der rückwirkungsfreie Betrieb der betreffenden Fahrzeugfunktion und aller betragenden Komponenten muss somit durch Integration geeigneter Sicherheitskonzepte garantiert werden. Hierfür kommen z.B. Isolationsmechanismen zum Einsatz, um potentielle Fehlerbilder von Drittanwendungen und anderen Störfaktoren abzuwehren.

Sogenannte Fehler gemeinsamer Ursache (*engl. Common Cause Failure (CCF)*) stellen laut Börcsök [45] Fehlerquellen dar, durch welche mehrere redundant ausgelegte Komponenten (Redundanz: Hardware, Software, Zeit und Information) gleichzeitig ausfallen. CCFs untergliedern Fehler, welche einen Ausfall bei mehreren Komponenten auf die gleiche Art und Weise hervorrufen (*engl. Common Mode Failure (CMF)*). Kaskadierte Ausfälle (*engl. Cascading Failures (CF)*) hingegen betreffen gezielt eine Komponente und haben keinen Einfluss auf redundante Komponenten. Alle aufgeführten Ausfallarten weisen ein statistisches Ausfallverhalten auf [105]). Weiterhin existieren Fehlerfälle, die den Ausfall des Gesamtsystems nach sich ziehen (*Single Point of Failure*).

Die ISO 26262 [89] schreibt vor, dass Rückwirkungsfreiheit zwischen Komponenten nur dann gilt, wenn kaskadierte Fehlerbilder ausgeschlossen werden können. Dies gilt einerseits für Sub-Elemente ohne ASIL, einem niedrigeren ASIL oder einem höheren ASIL, wenn dieser zu einer Verletzung eines Sicherheitszieles führt. Tritt also ein Fehler in einer benachbarten Komponente auf, muss garantiert werden, dass sicherheitsrelevante SWCs nicht in ihrer Funktionsweise beeinträchtigt werden oder gar indirekt zu einem Systemausfall führen. Abbildung 3.3 stellt hierzu den Ablauf einer möglichen Fehlerfortpflanzung über eine gemeinsam genutzte Hardwareressource dar [153]. Fehler können sich allerdings im Rahmen einer Wirkungskette auf abhängige Komponenten übertragen. Dabei wird von einer vernetzten, potentiell fehlerhaften Systemkomponente ausgegangen. Diese kann sowohl eine Softwarefunktionalität (z.B.

Bibliotheksfunktion) als auch ein Hardwarebauteil (z.B. gemeinsam genutzter Speicher) repräsentieren.

Weiterhin können Fehler auf informativer (z.B. Falschaussagen, Übertragungsprobleme) und zeitlicher Ebene (z.B. Deadlocks, falsche Ausführungsreihenfolge) hervorgerufen werden. Eine potentielle Fehlerfortpflanzung, ausgehend von nichtqualifizierten Komponenten, ist somit grundsätzlich möglich. Abhängigkeiten müssen daher durch technische Sicherheitskonzepte abgesichert oder die vernetzte Komponente selbst mit in die Betrachtung des technischen Sicherheitskonzepts aufgenommen werden. Damit wird allerdings der annotierte ASIL einer Komponente mit Sicherheitseinstufung vererbt [89].

3.4 Verlagerung von E/E-Funktionen

Sind Fahrzeugfunktionen nach ihren Domänen angeordnet, verlagern sich Software-Cluster zwischen Steuergeräten oder entfallen gänzlich. Der Anspruch den Wünschen der Kunden und ihren individuellen Fahrzeugkonfigurationen gerecht zu werden, resultiert in einer hohen Varianz an Fahrzeugausbaustufen. Softwareänderungen müssen über Produktreihen hinweg flexibel eingesteuert und unterschiedliche Ausbaustufen berücksichtigt werden können. Steuergeräte können somit in ihrem Umfang verringert bzw. gänzlich entfallen, wodurch Hardwarekosten eingespart werden können. Ein beispielhafter Anwendungsfall ist die optimierte Verteilung von Fahrzeugsoftware und -systemen zwischen ECUs.

Um die Variantenvielfalt einzuschränken wird die folgende Problemstellung ausschließlich in Voll- bzw. Grundausstattung von E/E-Umfängen eines Fahrzeuges reduziert. Unabhängig von der geometrischen Anordnung der Steuergeräte im Fahrzeug und der daran angeschlossenen Peripherie, müssen Applikationen softwareseitig verlagert werden. Im Fall einer Rationalisierung von Fahrzeugfunktionen müssen herstellerübergreifend vereinbarte Domänenschnittstellen zwischen DLRs für noch bestehende Anwendungen zur Verfügung stehen [17, 18]. Zusätzlich muss der Verbleib von OEM spezifischen Systemfunktionen (wie Betriebs-, Kunden-, Administrations- und Supportfunktionen) oder differenzierende Grundfunktionalitäten geregelt sein, die in jeder Ausbaustufe vorhanden sind.

3.4.1 Skalierbarkeitsanforderungen an zukünftige E/E-Systeme

Innerhalb dieser Arbeit wird unter der Begrifflichkeit Skalierbarkeit die flexiblen Konsolidierung bzw. Hochintegration zur freien Verteilung von Fahrzeugsoftware auf einem Steuergerät verstanden. Hierunter wird die flexible Integration von Softwarekomponenten bzw. -clustern versinnbildlicht. Diese müssen flexibel innerhalb Integrationssteuergeräten unterschiedlichen Rechenkernen zugewiesen werden können, ohne den bisherigen Betrieb zu beeinträchtigen.

Eine mögliche Verlagerung oder Migration von Softwareteilen eines Integrationssteuergerätes wird in Abbildung 3.4 dargestellt. Um Softwarekomponenten auf einer

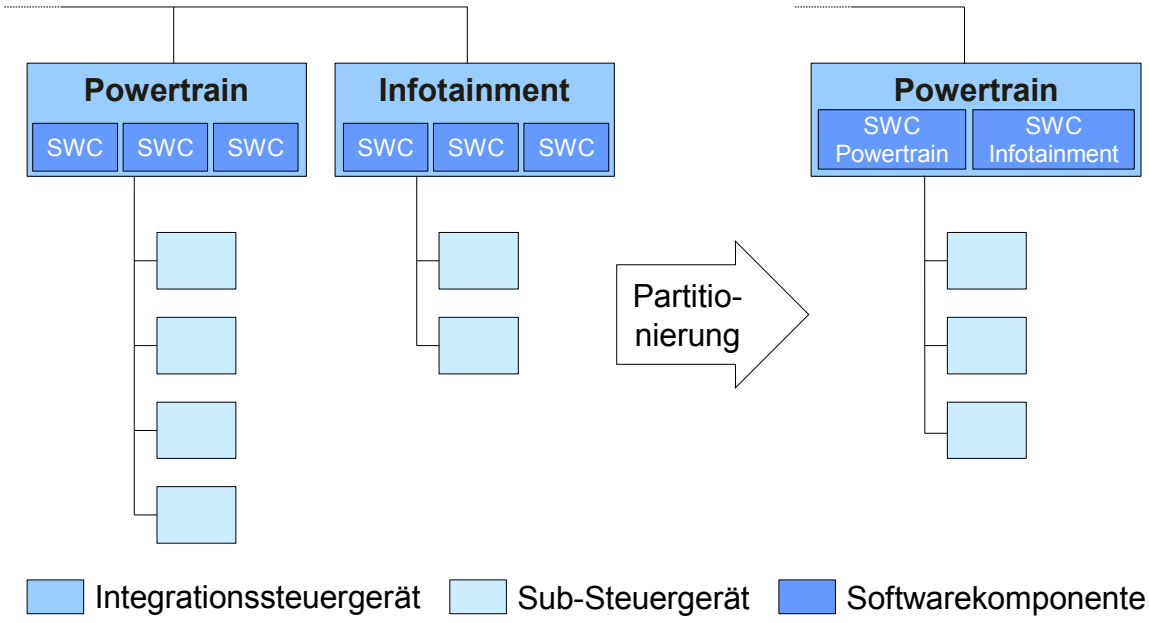


Abbildung 3.4: Partitionierung von Integrationssteuergeräten.

Plattform zu zentralisieren, wird in erster Linie in dieser Arbeit die Verlagerung und Partitionierung von Fahrzeugfunktionen und deren Softwarekomponenten untersucht. Der Parallelbetrieb von Applikationen wird dabei nicht behandelt.

Fahrzeugsoftware wird statisch vorkonfiguriert und im System integriert. Es wird von einer nahezu konstanten Systemlast ausgegangen, welche sich aus einer Vielfalt von zyklischen als auch vereinzelt sporadisch eintretenden Ereignissen zusammensetzt. Die Systemlast ist daher bis auf kleinere Schwankungen planbar. Aufgrund der statischen Integration wird der Speicher zur Integrationszeit zugewiesen. Der Bedarf steigt zur Laufzeit nicht dynamisch an. Der angewandte Programmablauf (*engl. Schedule*) wird im Voraus festgelegt und bleibt zur Laufzeit bestehen. Dabei stellen Mehrkernsysteme eine neue Herausforderungen dar.

Um eine deterministische Programmausführung nicht zu beeinflussen, werden Fahrzeugsoftwarekomponenten daher vollständig auf individuelle Rechenkerne partitioniert. Der Synchronisationsbedarf zwischen verlagerten Anwendungen soll minimalisiert sein, um zusätzlichen Rechenbedarf zu vermeiden. Werden zusätzliche Fahrzeugsoftwarekomponenten auf ein Echtzeitsystem integriert, müssen weiterhin alle Deadlines eingehalten werden können.

3.4.2 Identifizierte Integrationsszenarien

Zur Hochintegration von Fahrzeugfunktionen auf DLRs bzw. Integrationssteuergeräte werden drei potentielle Partitionierungsszenarien zur Evaluierung festgelegt. In Abbildung 3.5 sind jeweils zwei Domänenleitrechner (DLR1 und DLR2) und eine SUB-ECU (SUB1) abgebildet. Quadrate innerhalb der Rechner stellen exemplarisch zu verlagernde Fahrzeugfunktionen dar.

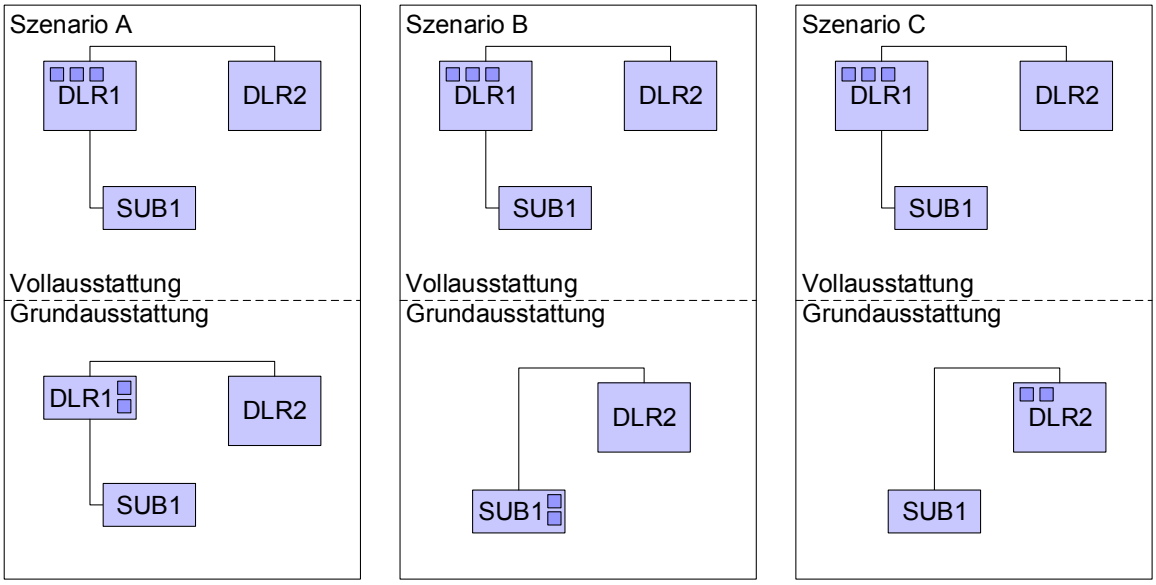


Abbildung 3.5: Szenarien zur Verlagerung und Partitionierung auf Bordnetzebene.

Grundsätzlich wird angenommen, dass jegliche Fahrzeugsoftware nach AUTOSAR-konformen Richtlinien entwickelt ist. Dabei beinhaltet DLR1 sicherheitsrelevante Softwareanteile, DLR2 hingegen ausschließlich nicht-sicherheitsrelevante Fahrzeugfunktionen. Die integrierte Basissoftware enthält daher keine Funktionalitäten zur Unterstützung der funktionalen Sicherheit.

Szenario A: Reduktion

In Szenario A wird angenommen, dass DLR1 bei niederen Ausbaustufen weniger Funktions- bzw. Schnittstellenumfänge beinhaltet. Wie in Abbildung 3.5 dargestellt, bleibt DLR1 als Steuergerät bestehen, verliert jedoch an seinem Funktionsumfang. Es wird angenommen, sobald Softwarekomponenten nicht mehr auf DLR1 integriert werden, der zu tätigende Konfigurationsaufwand innerhalb der Basissoftware gering ist. Dies setzt voraus, dass keine vernetzten Abhängigkeiten zu anderen Softwarekomponenten verletzt sind. Ist dies nicht der Fall, stellt dieses Szenario zum aktuellen Wissensstand keine weiteren Forschungsbedarf dar. Es soll daher ausschließlich als Vergleich mit den Folgeszenarien dienen.

Szenario B: Entfall

In Szenario B entfällt DLR1 vollständig als eigenständiges Steuergerät. Die verbleibenden Grundfunktionalitäten werden von SUB1 übernommen. Hierfür muss SUB1 genügend Hardwareressourcen, Rechen- und Speicherkapazitäten zur Verfügung stellen. SUB1 wird wie in Kapitel 3.3 erläutert von einem Zulieferer verantwortet. Dieses Szenario ist aufgrund verteilter Verantwortlichkeiten schwer zu realisieren. SUBs sind Zukaufteile und müssen daher austauschbar bleiben. Wird darin individuelle Software integriert, ginge diese Eigenschaft verloren. Weiterhin ist es fraglich, ob SUBs über

die nötigen Hardwareressourcen verfügen, um zusätzliche Rechen- und Speicherkapazitäten zu reservieren. Soll trotz dieser Einschränkungen Software verlagert werden, kann dies nur vereinzelt vollzogen werden.

Szenario C: Fusion

In Szenario C soll wiederum DLR1 vollständig entfallen. Die verbleibenden Grundfunktionalitäten müssen nun, wie in der rechten unteren Bildhälfte von Abbildung 3.5 erkennbar, mit DLR2 fusioniert werden. Wiederum müssen in DLR2 genügend Hardwareressourcen zur Verfügung stehen. DLR2 wird, wie in Kapitel 3.3 erläutert, vom OEM selbst verantwortet. Per Definition der Domänenleitrechner sollen diese als Integrationsplattform über die notwendigen Hardwareressourcen als Vorhalt verfügen. Es ist zudem anzunehmen, dass die benötigten Grundfunktionalitäten wie Controller für Hochgeschwindigkeitsbusse auf beiden ECUs vorhanden sind. Aufgrund der dargestellten Freiheitsgrade soll dieses Szenario im weiteren Verlauf der Arbeit zur Analyse verwendet werden.

3.5 Zusammenfassung des Kapitels

In diesem Kapitel wurde der Domänen-orientierte Architekturansatz eingeführt [121] und mit der aktuell vorherrschenden verteilten bzw. einer möglichen zentralisierten E/E-Architektur verglichen. Zur Reduktion der steigenden Anzahl von ECUs werden dabei zuvor verteilte Fahrzeuganwendungen auf gemeinsamen Integrationssteuergeräten konsolidiert. Dabei wurde aufgezeigt, dass der Domänen-orientierte Ansatz die Vorteile der verteilten und der zentralisierten Bordnetzarchitektur vereint.

Zur Segmentierung von verteilter Software auf Integrationssteuergeräte bzw. Domänenleitrechner wurde eine Herangehensweise vorgestellt, um Softwarecluster zu bilden. Ausgehend von einer logischen Abstraktionsebene werden dabei zusammengehörige Komponenten in einer technischen Ebene einzelnen Rechenkernen zugeordnet.

Zur Integration sicherheitsrelevanter Software müssen betreffende Komponenten durch geeignete Isolationsmechanismen voneinander abgeschottet werden. Dies ermöglicht einen rückwirkungsfreien Betrieb von sicherheitsrelevanten Fahrzeugfunktionen. Dabei wurde die Problematik aufgezeigt, dass auftretende Fehler sich im Gesamtsystem ausbreiten können. Sind also sicherheitsrelevante Komponenten in einem Steuergerät integriert, verlagert sich ihr jeweiliger ASIL unter Umständen auch auf verbundene Softwarekomponenten, welche dann gleichermaßen gemäß der ISO 26262 [89] qualifiziert werden müssen. Um den Aufwand zur Abnahme und Qualifizierung sicherheitsrelevanter Systeme einzugrenzen, sollte dies verhindert werden.

Es wurde weiterhin festgestellt, dass eine Zusammenfassung von ECUs erweiterte Schutzkonzepte zur Integration von sicherheitsrelevanten Fahrzeugfunktionen mit unterschiedlichen ASILs benötigt. Hierfür wurden die relevanten Anforderungen der ISO 26262 [89] aufgeführt und für weitere Integrationsschritte beurteilt [123]. Zusätzlich wurden potentielle Anforderungen für zukünftige Integrationssteuergeräte aufgeführt.

Zur Verlagerung von Fahrzeugsoftware wurden drei grundlegende Integrationsszenarien identifiziert (Reduktion, Wegfall, Fusion). Für den weiteren Verlauf der Arbeit soll die DLR-Fusion näher betrachtet werden.

4 Bewertung der Partitionierbarkeit sicherheitsrelevanter Softwarekomponenten

Ziel dieses Kapitels ist die Identifikation von architektonischen Schwachstellen in Softwarenetzen mit hohen Abhängigkeiten, unter Berücksichtigung sicherheitsrelevanter SWCs. Zur Hochintegration von DLRs (Szenario: DLR Fusion, dargestellt in Kapitel 3.4.2) sollen potentielle Problemstellungen bereits frühzeitig zur Entstehungsphase erkannt werden. Hierzu soll der Vernetzungsgrad von Softwarekomponenten, unter Verwendung von Techniken zur Analyse sozialer Netzwerke, quantifiziert und anhand einer Rangfolge gelistet werden. Dabei soll eine Aussage über den möglichen Grad der Partitionierbarkeit sicherheitsrelevanter Systeme gegeben werden. Dazu wird ein etabliertes, graphenbasiertes Verfahren zur Bewertung von Webseiten (Google's PageRank-Algorithmus, nach Brin and Page [49]) in einer neuen, modifizierten Form angewandt. Hierzu wird der Software Component Rank (SwcRank) eingeführt, um die Wichtigkeit vernetzter SWCs festzustellen. Die Durchgängigkeit der Methode wird anhand eines Fallbeispiels evaluiert.

4.1 Herausforderung und Problemstellung

E/E-Systeme im Fahrzeug weisen häufig eine **starke Vernetzung von SWCs** auf. Manche SWCs sind, z.B. aufgrund von Sicherheitseigenschaften, wichtiger als andere. Sie stellen möglicherweise einen Dienst für eine Vielzahl von Applikationen bereit oder implementieren OEM-spezifische Plattformfunktionalitäten (Diagnose, Kodierung). Aufgrund dieser Servicefunktionen sind viele SWCs in einer Kette von Abhängigkeiten zu weiteren Applikationen miteinander verbunden. Somit müssen zur Entwicklung der Softwarearchitektur eines Steuergerätes nicht-funktionale Anforderungen wie Sicherheit und Verlässlichkeit besonders berücksichtigt werden.

Durch den starken Vernetzungsgrad können für SWCs mit unterschiedlichen ASILs ungewünschte, jedoch nur schwer vermeidbare **Abhängigkeiten auf Applikationsebene** und in tieferliegenden Softwareschichten der AUTOSAR BSW entstehen. Um einen Zugriff auf die Hardware zu erleichtern, wird die AUTOSAR BSW meistens im Kernel-Modus betrieben. Zugleich sollen Basisfunktionalitäten aus Kostengründen von weiteren Qualifizierungsmaßnahmen hinsichtlich funktionaler Sicherheit ausgenommen werden.

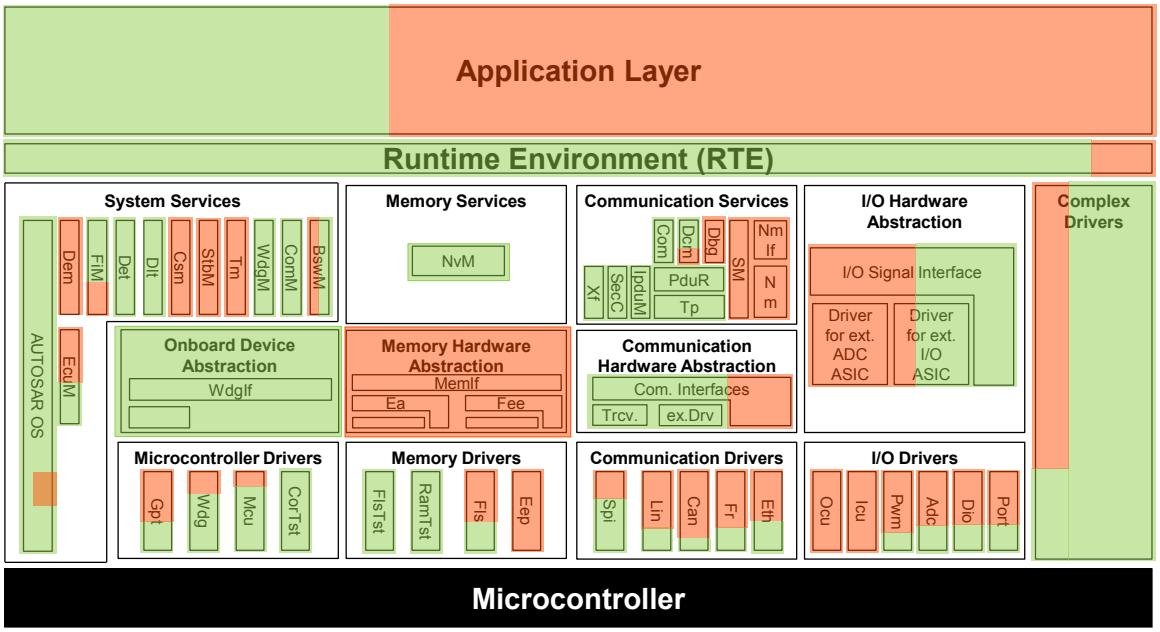


Abbildung 4.1: Vermaschung sicherheitsrelevanter Software (grün: trusted) und unqualifizierter Software (rot: untrusted) innerhalb der AUTOSAR BSW [152].

Jedoch wird innerhalb der ISO 26262 [89] die **Rückwirkungsfreiheit** zwischen Fahrzeugfunktionen gefordert. Dies impliziert eine vollständige Isolation von sicherheitsrelevanten Softwareanteilen. Das betrifft die gemeinsame Nutzung von Diensten und Softwarebibliotheken ohne Einbindung zusätzlicher Absicherungsmaßnahmen. Sind starke Abhängigkeiten zwischen der Anwendungsebene und der BSW vorhanden, muss die gesamte Software auf funktionale Sicherheit geprüft werden (siehe Kapitel 3.3.3).

Zetlmeisl et al. [152] beschreiben hierzu den Aufwand von häufigen Kontextwechseln zwischen unterschiedlichen Schutzleveln einer MPU und stellen die massiven **Abhängigkeiten innerhalb der AUTOSAR BSW** dar. Abbildung 4.1 zeigt hierzu ein Beispiel für den Betrieb von vertrauenswürdigen bzw. qualifizierten Anwendungen (grün) und nicht-qualifizierten, potentiell fehleranfälligen Softwareteilen (rot). Ein Beispiel für das Aufeinandertreffen unterschiedlicher ASILs sind Signale innerhalb des *I/O Hardware Abstraction* Blocks. Es existieren hierbei einerseits Signalein- bzw. ausgaben, welche sowohl für sicherheitsrelevante als auch nicht sicherheitsrelevante Fahrzeugfunktionen bestimmt sind. Die Konfiguration aller Signale findet in gemeinsamen BSW-Modulen und innerhalb der RTE statt und sollte rückwirkungsfrei voneinander betrieben werden können.

4.2 Verwandte Methoden und Techniken

Die amerikanischen Psychologen Milgram [109] und Travers et al. [146] zeigen auf, dass jede Person auf der Welt über gerundet 6 Bekanntschaften miteinander vernetzt

ist. Diese Tatsache wird als Kleine-Welt-Phänomen (*engl. Small World Phenomenon bzw. The Six Degrees of Separation*) bezeichnet. Diese Theorie kann auf verschiedene Problemstellungen transferiert werden [150]. Dies betrifft Durchsatzmessungen innerhalb von Computernetzwerken oder das Auffinden von möglichen Verwandtschaften innerhalb eines sozialen Netzwerkes. Somit kann auch der Vernetzungsgrad (*engl. node centrality*) bzw. die relative Wichtigkeit einzelner Knotenpunkte (z.B. bekannte Persönlichkeiten) unter Verwendung eines gerichteten Graphen ausfindig gemacht werden [42–44]. Freeman [57] beschreibt dies als *betweenness centrality*, um das Kommunikationsaufkommen über einen bestimmten Netzwerkknoten zu errechnen.

Zur Erzielung einer möglichst hohen Kohäsion laut Parnas [116] und Stevens et al. [139], sollen Softwarekomponenten jeweils einer individuell spezifizierten Aufgabe zugeordnet werden (*engl. separation of concerns*). Wegen komplexer und veralteter Softwarestrukturen können Fahrzeuganwendungen meist nicht komponentenweise aufgespalten werden. Querschnittliche Belange (*engl. cross cutting concerns*) erschweren bzw. verhindern zusätzlich eine klare Partitionierung von Fahrzeugfunktionen (z.B. funktionale Sicherheit, Performanz- oder Hochverfügbarkeitsansprüche). Nicht-funktionale Anforderungen sind mit einer Vielzahl von Softwareteilen und -komponenten verwoben und können meist schwer individuell beurteilt werden.

Fahrzeugsoftwaresysteme verwenden meist eine gemeinsame BSW und integrieren eine Vielzahl von SWCs. Innerhalb eingebetteter Mehrkernsysteme ist das Wissen über SWCs, welche als Kommunikationsknotenpunkt fungieren, ein Anhaltspunkt, um eine optimale Verteilung von Applikationen auf Rechenkerne vornehmen zu können [36, 51]. Diese Problematik betrifft auch aktuelle eingebettete Echtzeitsysteme im Automobilsektor [132, 135].

Inoue et al. [85] verwenden bereits Methoden zur Analyse sozialer Netzwerke (wie den PageRank Algorithmus [115]), um die Wiederverwendbarkeit von Funktionen in großen Software-Frameworks zu identifizieren. Für Frameworks eingebetteter E/E-Systeme wird dieser Ansatz von Hobelsberger [82] erweitert und angewandt.

4.3 Der PageRank

Der PageRank Algorithmus ist eine Methode, um die Wichtigkeit (*engl. centrality*) von Webseiten zu berechnen [49, 115]. Diese Methode wird vor allem von der Suchmaschine Google angewandt. PageRank ist eine bekannte Methode, um ein Ranking zwischen Webseiten zu erstellen und basiert auf deren Verlinkungsstruktur. Das Vorgehen ist ähnlich zum Hypertext-Induced Topic Selection (HITS) Algorithmus, welcher von [96] entwickelt wurde. Neben dem PageRank und HITS Algorithmus existieren viele andere Methoden zur Feststellung der Wichtigkeit von Komponenten innerhalb eines Graphen, wie *Betweenness-*, *Degree-* and *Closeness centrality* [57, 150].

Der PageRank Algorithmus basiert auf einem Eigenvektorproblem [100]. Hieraus resultiert die relative Wichtigkeit einer Webseite im Vergleich zu allen anderen Webseiten im World Wide Web. Der Rang einer Webseite basiert auf dessen Vorgängern, welche auf diese Webseite verweisen (*engl. backlinks*). Die Wichtigkeit dieser Webseiten geht dadurch auf deren Nachfolger über (gedämpft durch den Faktor α). Websei-

ten sind wichtiger, sobald viele wichtige Webseiten ebenfalls auf diese verweisen. Es handelt sich um einen rekursiven Algorithmus, wobei der Rang einer Webseite individuell kalkuliert wird. Hierfür sind eine Vielzahl von Iterationsschritten notwendig, um Konvergenz herzustellen und ein genaues Resultat zu erzielen.

Im Folgenden wird der PageRank Algorithmus aufgeführt. Es werden N Knoten mit den Rängen $R_i, i = 1, \ldots, N$, mit $N \in \mathbb{N}$ betrachtet. Ist ein Knoten $j \in \{1, \ldots, N\}$ ein Vorgänger von i, so wird dieser durch $j \to i$ dargestellt. Die Anzahl ausgehender Links eines Knotens j wird als Grad ausgehender Kanten $deg^-(j)$ (*engl. outdegree*) bezeichnet. Der normalisierte PageRank Algorithmus für Rang R_i lautet:

$$R_i = \alpha \sum_{j \to i} \frac{R_j}{deg^-(j)} + \frac{1-\alpha}{N}, \tag{4.1}$$

wobei der Parameter $\alpha \in [0, 1]$ als Dämpfungsfaktor dient (z.B. $\alpha = 0.85$). Dieser Faktor kann frei gewählt werden und beeinflusst die Konvergenzgeschwindigkeit des Algorithmus. Die zugehörige Matrix-Vektor-Gleichung lautet

$$\vec{R} = \alpha M \vec{R} + \frac{1-\alpha}{N} \vec{1}, \tag{4.2}$$

wobei $\vec{1} = (1, \ldots, 1)^T \in \mathbb{R}^N$ einen Vektor repräsentiert, in welchem alle Einträge gleich 1 sind. Weiterhin ist die Übergangsmatrix M definiert durch:

$$M_{i,j} = \begin{cases} \frac{1}{deg^-(j)} & \text{wenn } j \to i, \\ 0 & \text{sonst.} \end{cases} \tag{4.3}$$

.

4.4 Der TrustRank

Für Internet-Suchmaschinen ist die Unterscheidung zwischen wichtigen Webseiten und unwichtigen Webseiten essentiell. Es existieren vermeintlich wichtige Webseiten, welche lediglich stark vernetzt sind um ihren PageRank zu erhöhen [49]. Um diesem Problem entgegenzuwirken, wurde der Algorithmus von Google Inc. erweitert, woraus der TrustRank nach Gyongyi et al. [70] entstanden ist. Dabei werden Webseiten zusätzlich nach deren Vertrauenswürdigkeit eingestuft, um somit die Qualität der vorherigen PageRank-Berechnung zu verbessern.

Der *inverse* PageRank Algorithmus ermittelt Webseiten, welche als mögliche *Seeds* verwendet werden können. *Seeds* stellen die potentiell wichtigsten Webseiten dar und geben ihr Vertrauen (*engl. Trust*) an viele andere verlinkte Webseiten weiter. Im besten Fall verweisen diese wiederum auf Webseiten, welche gleichwertig oder höher eingestuft sind.

Zur Berechnung des *inversen* PageRanks fungiert ein Knoten i als potentieller *Seed*. Dies wird definiert durch:

$$R_i = \alpha \sum_{i \to j} \frac{R_j}{deg^+(j)} + \frac{1-\alpha}{N}, \tag{4.4}$$

wobei die Anzahl eingehender Links eines Knotens j anhand des Grades eingehender Kanten (*engl. indegree*) mit $deg^+(j)$ bezeichnet wird. Die äquivalente Matrix-Vektor-Gleichung lautet:

$$\vec{R} = \alpha \widetilde{M} \vec{R} + \frac{1-\alpha}{N} \vec{1}. \tag{4.5}$$

Die Übergangsmatrix $\widetilde{M}$ ist definiert durch:

$$\widetilde{M}_{i,j} = \begin{cases} \frac{1}{deg^+(j)} & \text{wenn } i \to j, \\ 0 & \text{sonst.} \end{cases} \tag{4.6}$$

Der Rang der Knoten, welche vertrauenswürdig und gleichzeitig eine *Seed*-Webseite sind, wird in $\vec{d}$ gespeichert. Die verbleibenden Einträge des Vektors werden zu 0 gesetzt. Danach wird $\vec{d}$ auf 1 normalisiert.

Nun wird ein ähnlicher Schritt zu (4.1) berechnet. Dieser repräsentiert eine verzerrte (*engl. biased*) Version des normalisierten PageRank Algorithmus, dessen Summe aller Knotenränge auch 1 ergibt. Der TrustRank Algorithmus für R_i wird definiert durch

$$R_i = \alpha \sum_{j \to i} \frac{R_j}{deg^-(j)} + (1-\alpha) d_i, \tag{4.7}$$

wobei d_i das jeweilige Gewicht, versinnbildlicht durch Seriosität, eines Knotens i darstellt. Die zugehörige Matrix-Vektor-Gleichung lautet

$$\vec{R} = \alpha M \vec{R} + (1-\alpha)\vec{d}, \tag{4.8}$$

wobei $d = (d_1, \ldots, d_N)^T \in \mathbb{R}^N$ und die Übergangsmatrix M ist identisch zur Übergangsmatrix des PageRank Algorithmus, vgl. (4.3).

4.5 Der Software Component Rank

Fahrzeugsoftware ist sowohl innerhalb als auch über Steuergerätegrenzen hinweg meist weitverzweigt vernetzt. Um die relative Wichtigkeit einzelner SWCs innerhalb sicherheitsrelevanter Fahrzeugsoftwaresysteme feststellen zu können, sollen Ansätze zur Analyse sozialer Netzwerke angewandt werden. Zu diesem Zweck wird in dieser Arbeit der *Software Component Rank* entwickelt und eingeführt. Dieser erstellt eine Rangfolge zwischen vernetzten SWCs und repräsentiert eine Modifikation des TrustRank Algorithmus [120].

Das Netzwerk einzelner Softwarekomponenten kann, ähnlich zum PageRank Algorithmus, als gerichteter Graph beschrieben werden. Auch hierbei sind alle Einträge innerhalb der resultierenden Adjazenzmatrix durchgehend positiv. Softwareeinheiten besitzen darin eine unterschiedliche Anzahl von Eingangs- bzw. Ausgangssignalen. Jedoch ist in den wenigsten Fällen ein Softwarenetzwerk stark zusammenhängend, in welchem gerichtete Kanten alle Knoten geschlossen miteinander verbinden.

Die Granularität der Softwareeinheiten bzw. Knoten im Graphen wird hierfür nicht festgelegt. Knoten werden als logisch zusammenhängende Container verstanden und können:

- auf kleinster Ebene einzelne C-Funktionen,
- logische Container als atomare AUTOSAR Softwarekomponenten,
- Kompositionen aus Gruppierungen von Anwendungen,
- oder gar vollständige ECUs

darstellen.

4.5.1 Bewertung von Fahrzeugsoftwarekomponenten

Ziel des SwcRanks ist die Identifizierung zentraler Knotenpunkte (*engl. hot spots*) innerhalb des Softwarenetzwerkes. Identisch zum PageRank (siehe Kapitel 4.3) orientiert sich der SwcRank unter anderem an der Vernetzungsstruktur referenzierender Elternknoten (*engl. backlinks*). Eine Softwarekomponente ist somit wichtiger, sobald viele Komponenten auf diese referenzieren. Der erzielte Grad an Wichtigkeit wird hierbei von den Elternknoten auf die einzelne Softwarekomponente übertragen.

Eine Softwarekomponente kann weiterhin eine Vielzahl nachfolgender Komponenten referenzieren (*engl. frontlinks*). Der Rang referenzierter Softwarekomponenten fließt anteilig zurück und erhöht somit die Wichtigkeit der ausgehenden Komponente.

Wir betrachten ein Netzwerk bestehend aus N Softwarekomponenten mit den Rängen R_i, wobei $i \in \{1, \ldots, N\}$. Der SwcRank in Matrix-Vektorschreibweise wird folgendermaßen dargestellt:

$$\vec{R} = \alpha A \vec{R} + \frac{(1-\alpha)}{N} \vec{1}. \tag{4.9}$$

Dabei stellt A die Übergangs- bzw. Adjazenzmatrix des Graphen dar. Der Dämpfungsfaktor $\alpha \in [0, 1]$ ist gewählt bei Page et al. [115]. Er fungiert als globaler Designparameter und steuert die Verbreitung der Knotengewichte.

Analyse der Backlinks

Innerhalb der ersten Variante des Algorithmus wird die Wichtigkeit einer Komponente anhand ihrer Backlinks ermittelt. Die Anzahl eingehender Verbindungen soll hierfür analysiert werden. Dabei ruft eine Softwarekomponente j die Komponente i auf ($j \rightarrow i$) bzw. sendet Signale zu dieser. Die Übergangsmatrix definiert durch $A = M$, wobei gilt

$$M_{i,j} = \begin{cases} 1 & \text{falls } j \rightarrow i, \\ 0 & \text{sonst.} \end{cases} \tag{4.10}$$

Analyse der Frontlinks

Die zweite Variante zur Analyse der Frontlinks orientiert sich am inversen PageRank bzw. TrustRank nach Gyongyi et al. [70]. Die Anzahl ausgehender Verbindungen soll hierfür analysiert werden. Der Rang eines Knotens wird somit anhand seiner

Frontlinks bewertet. Dabei ruft Softwarekomponente i eine weitere Komponente j auf bzw. sendet Signale zu dieser. Zur Rückwärtsverlinkung wird zur Kalkulation eine transponierte Matrix M^T verwendet, diese lautet

$$M_{i,j}^T = \begin{cases} 1 & \text{falls } i \to j, \\ 0 & \text{sonst.} \end{cases} \tag{4.11}$$

Dieser Vorgang ist ähnlich zur Berechnung von *Seeds* innerhalb des TrustRank Algorithmus (siehe Kapitel 4.4). Jedoch ist die inverse Übergangsmatrix des SwcRanks nicht stochastisch.

Vergleich mit verwandten Lösungsverfahren

Verbindungen zwischen E/E-Softwareeinheiten innerhalb des SwcRanks sind statisch und nicht stochastisch. Die resultierende Adjazenzmatrix enthält somit zu Beginn lediglich binäre Einträge. Der nicht normalisierte Bewertungsvektor stellt den entscheidenden Unterschied zwischen dem klassischen PageRank Algorithmus dar, in welchen jede Spalte der Matrix aufsummiert 1 ergibt (links-stochastisch). Dies stellt ein ähnliches Verfahren zum HITS Algorithmus dar, welcher von Kleinberg [96] entwickelt wurde. Somit hat die Anzahl der Beziehungen zwischen einzelnen SWCs keinen Einfluss auf die Rangverteilung.

Das *Random-Surfer-Model* des PageRank Algorithmus stellt die jeweilige Übergangswahrscheinlichkeit dar, mit welcher ein Internet-Surfer zufällig auf eine Verlinkung zu einer Webseite klickt [49, 115]. Dieser stochastische Ansatz lässt sich auf ein eingebettetes Echtzeitsystem nicht übertragen, da dieses statisch vorkonfiguriert ist und nach einem striktem Ablaufplan arbeitet. Die Gewichtung einer Softwarekomponente wird somit nicht durch die Anzahl benachbarter Knoten aufgeteilt.

Hierzu sei erwähnt, dass der Front- bzw. Backlink-orientierte Ansatz ausschließlich für gerichtete Graphen zu unterschiedlichen Ergebnissen führen. Liegt ein symmetrischer bzw. ungerichteter Graph vor, sind beide Ergebnisse identisch.

Iterativer Lösungsansatz

Für beide genannten Ansätze wird (4.9) iterativ aufgelöst. Dabei stellt $\vec{R}_0$ einen willkürlich gewählten Startvektor dar. Dieser kann zu Beginn auch identische Werte enthalten. Im k-ten Iterationsschritt wird

$$\vec{r}^{(k)} = \alpha A \vec{R}^{(k-1)} + \frac{(1-\alpha)}{N}\vec{1}, \tag{4.12}$$

$$\vec{R}^{(k)} = \frac{1}{\|\vec{r}^{(k)}\|_1}\vec{r}^{(k)} \tag{4.13}$$

berechnet, wobei $\|\vec{r}^{(k)}\|_1$ die ℓ^1-Norm bezeichnet. Nach jedem Berechnungsdurchlauf werden jegliche Ergebnisse normalisiert. Der Vektor $\vec{R}^{(k)}$ hat für alle $k \geq 1$ stets die Norm 1. Der Parameter α regelt wiederum die Konvergenzgeschwindigkeit.

Letztlich resultieren zwei individuelle Ergebnisse: alle Werte des Frontlink- bzw. Backlink-orientierten Ansatzes. Jegliche Wichtigkeiten können somit einzeln analysiert werden. Damit können genau jene Komponenten gefiltert werden, welche entweder viele Informationen erhalten oder zur Verfügung stellen. Die Intensität des Kommunikationsaufkommens der SWCs innerhalb eingebetteter Systeme ist unterschiedlich hoch. Einige von ihnen nehmen mehr Informationen auf, andere wiederum generieren Daten für eine Vielzahl nachfolgender Komponenten. Aufgrund dieses Ungleichgewichts wird empfohlen, die Ergebnisse des Front- und Backlink Ansatzes zu kombinieren. Um hieraus eine finale Aussage der Zentralität einzelner Knoten bilden zu können, werden beide Ergebniswerte addiert.

4.5.2 Komplexität und Konvergenz

Die Adjazenzmatrix verbundener SWCs ist dünn besetzt, da die meisten Knoten lediglich mit einer kleinen Anzahl weiterer Knoten verbunden sind [100]. Ähnlich zum PageRank Algorithmus ist der Rechenaufwand des SwcRanks $\mathcal{O}(K * N)$. Die Berechnung hängt von der absoluten Anzahl aller Knoten N und der durchgeführten Anzahl von Iterationsschritten K ab.

Um Konvergenz sicherzustellen, wird der Vektor nach jeder vollständigen Iteration normalisiert (siehe Gleichung (4.12) und (4.13)). Dies verhindert eine stetige Erhöhung der Bewertungsergebnisse nach jeder Iteration. Das dargestellte Vorgehen ähnelt dem HITS-Algorithmus [96].

Der finale SwcRank Vektor zur Lösung von (4.9) wird iterativ berechnet. Die Berechnung kann abgebrochen werden, sobald ein Konvergenzkriterium erfüllt ist. Dies tritt ein, sobald der Unterschied zwischen der aktuell berechneten Knotenwichtigkeit und dem vorherigen Ergebnis hinreichend klein ist. Hierzu wird das Abbruchkriterium ϵ definiert. Die Berechnung wird beendet, wenn

$$|\vec{R}_i^{(k)} - \vec{R}_i^{(k-1)}| \leq \epsilon \tag{4.14}$$

für jeden Knoten $\vec{R}_i$ mit $i \in \{1, ..., N\}$ innerhalb der k-ten berechneten Iteration *wahr* ist. Die Berechnungsdauer des Algorithmus richtet sich nach der Größe des Abbruchkriteriums ϵ. Wird ein kleineres ϵ gewählt, werden genauere Ergebnisse erzielt. Jedoch sind hierfür zusätzliche Iterationsschritte notwendig.

4.5.3 Berücksichtigung sicherheitsrelevanter Softwarekomponenten

Funktionale Sicherheit für Verkehrsteilnehmer (wie Fahrzeuginsassen) ist eine wichtige nicht-funktionale Anforderung für OEMs. Innerhalb von E/E-Systemen müssen sicherheitsrelevante Fahrzeugfunktionen und deren Daten vor Fremdeinwirkung geschützt werden. Gemäß ISO 26262 [89] können Softwarefehler örtlich, zeitlich und während des Informationsflusses auftreten. Der AUTOSAR Standard spezifiziert diverse Schutzmechanismen, um Rückwirkungsfreiheit zwischen Softwarekomponenten

garantieren zu können. Innerhalb des SwcRanks werden sicherheitsrelevante Softwarekomponenten berücksichtigt, indem diese ein zusätzliches Gewicht erhalten. Die Iteration

$$R_i = \alpha \sum_{j \to i} R_j + (1 - \alpha) d_i, \tag{4.15}$$

verwendet einen Vektor $\vec{d} = (d_1, \ldots, d_N)$, um zusätzliche Gewichte für sicherheitsrelevante Knoten realisieren zu können. Im Folgenden wird eine mögliche Wahl für $\vec{d}$ dargestellt. Analog hierzu lautet die Matrix-Vektor-Gleichung

$$\vec{R} = \alpha M \vec{R} + (1 - \alpha)\vec{d}, \tag{4.16}$$

in welcher die Übergangsmatrix M im Falle eines Frontlink basierten Ansatzes durch M^T ersetzt werden muss. Die zugehörige Übergangsmatrix aus (4.10) und (4.11) kann individuell für den Front- bzw. Backlink Ansatz verwendet werden.

Die größte Neuerung des SwcRanks ist die Verwendung von konstanten Sicherheitseinstufungen, welche in $\vec{d}$ vorbelegt werden. Jedes Element, welches Sicherheitsanforderungen umsetzt, wird mit einem zuvor festgelegten ASIL versehen.

Wie in ISO 26262 [89] stellen $\sigma_A < \sigma_B < \sigma_C < \sigma_D$ die Gewichte der ASILs A, B, C und D dar. Diese können frei, je nach Gewichtung des jeweiligen ASILs, vergeben werden und setzen ingenieurmäßiges Vorwissen voraus. Ähnlich zum TrustRank Algorithmus wird eine Kategorisierung gemäß ISO 26262 [89] zwischen den unterschiedlichen ASILs T_A, T_B, T_C und T_D eingeführt. Exemplarisch stellt T_A eine Menge an Knoten dar, welche dem ASIL A zugeordnet werden kann.

Wir definieren für $i \in \{1, \ldots, N\}$

$$d_i = \begin{cases} \sigma_A & \text{wenn } i \in T_A, \\ \sigma_B & \text{wenn } i \in T_B, \\ \sigma_C & \text{wenn } i \in T_C, \\ \sigma_D & \text{wenn } i \in T_D, \\ 0 & \text{sonst.} \end{cases} \tag{4.17}$$

Ist $\vec{d} = \vec{0}$, hat die Gleichung $\vec{R} = \alpha M \vec{R}$ die nicht-triviale Lösung $\vec{R}$, ausschließlich wenn $1/\alpha$ ein Eigenwert von M ist. Da M nur endlich viele Eigenwerte besitzt, ist $\frac{1}{\alpha}$ in den meisten Fällen kein Eigenwert. Deshalb ist es relevant, dass $\vec{d}$ mit mindestens einem positiven Wert vorbelegt ist. Andernfalls ist das lineare Gleichungssystem homogen und hat die triviale Lösung (alle Werte sind 0).

Ein Beispiel des SwcRanks wird in Abbildung 4.2 dargestellt. Darin wird ein gerichteter Graph bestehend aus sechs Knoten gezeigt. Knoten 2 wird eine zusätzliche Sicherheitseinstufung zugeordnet. Beispielhaft wird $\alpha = 0.85$ und $\vec{d} = \{0, 0.5, 0, 0, 0, 0\}$ parametriert. Der Startvektor wird $\vec{R}_0 = \vec{1}$ gesetzt. Die Farbintensität in Abbildung 4.2 repräsentiert die Vererbung der Wichtigkeit bzw. Zentralität einzelner Knoten. Da Knoten 2 ein zusätzliches Gewicht zugewiesen wird, ist dessen Rang höher als die Ränge der benachbarten Knoten.

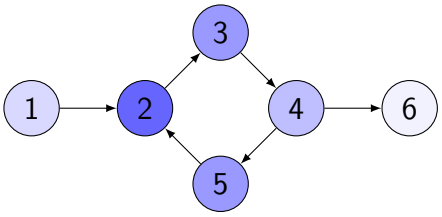


Abbildung 4.2: Beispielhafter Graph des Software Component Ranks, wobei die Gewichtung der Knoten durch die Farbintensität ausgedrückt ist.

Knoten-ID	**PageRank**	**SwcRank**	**Backlink Ansatz**	**Frontlink Ansatz**
1	0.046	0.208	0	0.208
2	0.208	0.466	0.238	0.228
3	0.222	0.398	0.225	0.173
4	0.234	0.373	0.190	0.183
5	0.145	0.381	0.173	0.208
6	0.145	0.173	0.173	0

Tabelle 4.1: Vergleich der Wichtigkeiten bzw. Zentralität einzelner Softwareknoten.

Tabelle 4.1 zeigt die ermittelten Ergebnisse aus Abbildung 4.2 im Vergleich zum PageRank Algorithmus [120]. Der Rang von Knoten 2 wird auf benachbarte Komponenten aufgeteilt. Knoten 2-5 bilden einen Ring, wodurch sich die Wichtigkeit zwischen diesen Softwarekomponenten stärker vererbt. Knoten 1 erbt Wichtigkeit von Knoten 2 durch den Frontlink-orientierten Ansatz. Dadurch nimmt die Wichtigkeit von Softwarekomponenten zu, sobald diese auf weitere wichtige Knoten referenzieren. Die Vererbung der Wichtigkeit wird bis hin zu Knoten 6 gedämpft. Da Knoten 1 keine Backlinks und Knoten 6 keine Frontlinks aufweisen, ist deren resultierende Wichtigkeit beim Backlink- bzw. Frontlink-Ansatz gleich null (siehe Spalte 4 und 5).

4.5.4 Gewichtete Beziehungen zwischen Softwarekomponenten

Neben einzelnen Verbindungen können auch mehrfache Verbindungen zwischen SWCs aufgebaut werden. Darüber hinaus werden Signale innerhalb eines gewissen Zeitfensters zwischen SWCs mit einer unterschiedlichen Frequenz übertragen. Aufgrund dieser zusätzlichen Einschränkungen werden die Kanten innerhalb des SwcRank-Graphen mit zusätzlichen Gewichten versehen.

Zur Backlink-Analyse ist $N_{i,j}$ die Anzahl der Verbindungen ausgehend von j, welche auf i referenzieren und $\omega_{i,j}^{l}$ das Gewicht der l-ten Verbindung, $l = 1, ..., N_{i,j}$. Die Einträge der Übergangsmatrix M werden

$$M_{i,j} = \begin{cases} \sum_{l=1}^{N_{i,j}} \omega_{i,j}^{l} & \text{falls } j \rightarrow i, \\ 0 & \text{sonst,} \end{cases} \tag{4.18}$$

gesetzt. Die Definition von M^T ist wie folgt: Frontlinks werden anstelle der Backlinks eingesetzt und die Gewichte aller ausgehenden Kanten eines Knotens werden aufsummiert.

Im Rahmen dieser Arbeit werden folgende drei Granularitätsstufen definiert, um Kantengewichte einzuführen:

1. Relationen zwischen SWCs repräsentieren das grundlegende Kommunikationsverhalten.

2. Die Multiplizität einer Verbindung beschreibt die Anzahl der Abhängigkeiten zwischen SWCs.

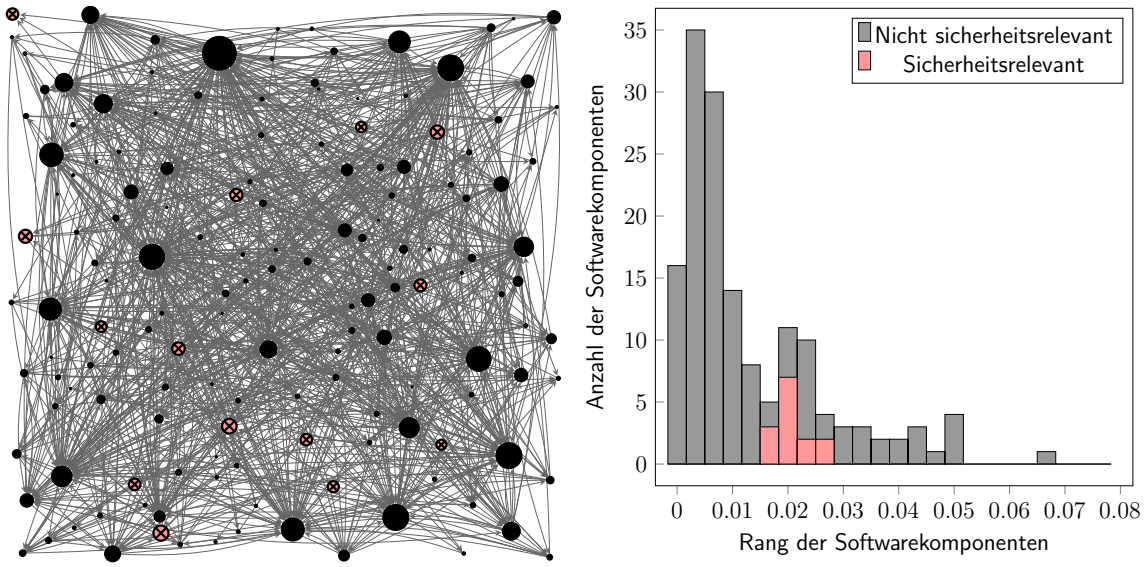
3. Die Frequenz gibt Aufschluss, wie oft SWCs über eine einzelne Abhängigkeit bzw. Verbindung Nachrichten miteinander austauschen.

Basierend auf der *AUTOSAR Top-Level Architektur* können die ersten beiden Stufen extrahiert werden [15]. Die dritte Abstufung kann durch extern hinzugefügte Informationen (wie Wiederholungsraten der Tasks) angereichert werden [27]. Sicherheitsrelevante Knoten müssen weiterhin manuell annotiert werden. Unter Verwendung des SwcRanks, wird zur Erfassung aller auftretenden Abhängigkeiten empfohlen, die Gesamtanalyse eines Steuergerätes sollte auf C-Funktionsebene durchzuführen. Für die AUTOSAR BSW liegt hierbei keine vollständig abstrahierte Beschreibung vor.

4.6 Analyse von ECU internen Softwareabhängigkeiten

Der aufgeführte Algorithmus soll nun anhand eines Fallbeispiels evaluiert werden. Als Evaluationsziel sollen wichtige Komponenten gefiltert und der Einfluss von sicherheitsrelevanter Software dargestellt werden. Hierfür wird ein exemplarischer Kommunikationsgraph eines Integrationssteuergerätes (siehe Kapitel 3.3) verwendet, welches unter anderem die Lichtansteuerung der Fahrzeugscheinwerfer umsetzt. Darin bilden verbundene SWCs ein Netzwerk. Der Kommunikationsfluss verketteter Komponenten ist entweder zyklisch oder sporadisch.

Abbildung 4.3a stellt die TopLevelComposition der Softwarearchitektur dar [120]. Jeder Knoten stellt eine SWC dar. Sicherheitsrelevante SWCs sind rot gefärbt und mit einem Kreuz versehen. Der Graph beinhaltet über 150 Knoten, verbunden mit mehr als 1000 Kanten (z.B. *AUTOSAR Assembly Connectoren*). SWCs können mehrfach miteinander verbunden sein. Der Graph beinhaltet noch keine spezifischen Kantengewichte. Sicherheitsrelevante SWCs werden mit ASIL B annotiert und implementieren Redundanz- und Validierungsfunktionen. Um Qualifizierungskosten zu senken (keine Vererbung des ASILs) und Rückwirkungsfreiheit sichern zu können, sollen Abhängigkeiten zwischen sicherheitsrelevanten und unqualifizierten SWCs reduziert werden.



(a) Ergebnisgraph des SwcRanks.

(b) Verteilung der Wichtigkeiten.

Abbildung 4.3: Darstellung des Softwarenetzes eines Integrationssteuergerätes inklusive deren sicherheitsrelevanter Softwarekomponenten [120].

Zur Berechnung der Wichtigkeit aller SWCs in Abbildung 4.3b werden folgende Parameter gewählt. Wie von Page et al. [115] empfohlen, wird der Dämpfungsfaktor $\alpha = 0.85$ gesetzt. Die Gewichte der ASILs in $\vec{d}$ werden wiederum linear verteilt (siehe Kapitel 4.5.3), wobei $\sigma_A = 0.25$, $\sigma_B = 0.5$, $\sigma_C = 0.75$ und $\sigma_D = 1$. Verbleibende Komponenten werden $d_i = 0$ gesetzt. Als Startvektor wird $\vec{R}_0 = \vec{1}$ gewählt, das Abbruchkriterium für dieses Beispiel lautet $\epsilon = 1\text{e}{-6}$ (siehe Kapitel 4.5.2).

Durch Wahl von ϵ wird die Berechnung bereits nach 13 Schritten beendet. Die Messung kann durch Veränderung des Faktors α beeinflusst werden. Wird ein Wert nahe 1 gewählt, werden Wichtigkeiten stärker im Graphen verteilt. Bei Wahl eines kleineren Wertes ist die Verbreitung von Wichtigkeiten restriktiver, wodurch das Abbruchkriterium noch schneller erreicht werden kann. Somit hat α einen starken Einfluss zur Glättung der Wichtigkeiten innerhalb des Graphen. Sollen hingegen sicherheitsrelevante Komponenten stärker berücksichtigt werden, muss alpha kleiner gewählt werden, sodass der additive Term $\vec{d}$ in (4.16) stärker gewichtet wird. Praktisch gesehen kann $1 - \alpha$ als Wahrscheinlichkeit für azyklische Aufrufe im Falle von ereignisorientierten Beziehungen dienen.

Liegt ein großer Graph vor, müssen aufgrund der Übersichtlichkeit numerische Werte graphisch analysiert werden. Abbildung 4.3b zeigt die Rangverteilung des Graphen. Die meisten Komponenten besitzen einen Rang kleiner 0.02, wobei sich die meisten Komponenten im Bereich 0.0033 bündeln. Sicherheitsrelevante SWCs verteilen sich im Bereich 0.02. Es existieren einzelne Hot Spots mit einem größeren Rang zwischen 0.04 und 0.07. Daraus folgt, dass es lediglich wenige SWCs gibt, die stark im Graphen vernetzt sind.

Im aktuellen Beispiel, illustriert in Abbildung 4.3a, sind lediglich AUTOSAR SWCs miteinander vernetzt. Werden diese mit weiteren BSW Modulen verbunden, muss auch deren Abhängigkeit zu sicherheitsrelevanten Komponenten geprüft werden. Die Analyse auf Funktionsebene bietet hierfür eine zusätzliche Detailstufe an. Die größten Knoten in Abbildung 4.3 stellen BSW Module bzw. Servicekomponenten dar. Beispielsweise ist der sogenannte *Diagnostic Event Manager* zur Nachverfolgung von *Diagnostic Trouble Codes* eine stark vernetzte Komponente [24]. Diese ist einmalig innerhalb einer ECU integriert und besitzt somit Abhängigkeiten zu nahezu allen SWCs. Hierbei muss klar unterschieden werden, mit welcher Frequenz SWCs Nachrichten mit solchen Komponenten austauschen. Servicefunktionen sind meist hoch vernetzt, tauschen allerdings nur selten (ausschließlich auf Anfrage) Informationen aus.

Um einen gewissen Grad an Isolation zwischen sicherheitsrelevanten SWCs und anderen SWCs garantieren zu können, besitzen diese Komponenten wenige Abhängigkeiten zu anderen. Je nach Wahl des Faktors α hat die Wichtigkeit einer Komponente Einfluss auf benachbarte SWCs. Dies gilt auch für Module der AUTOSAR BSW, bei welchen Isolationsmechanismen angewandt werden müssen. Ist dies nicht möglich, müssen diese entweder vollständig partitioniert, oder nach dem jeweilig höchsten ASIL der verbundenen Fahrzeugfunktion qualifiziert werden.

4.7 Bewertung und Einsatzgebiete

Sicherheitsrelevante Fahrzeugfunktionen beinhalten meist eine Vielzahl vernetzter SWCs. Weitverzweigte Abhängigkeiten zu anderen Funktionen können unter Umständen zusätzliche Qualifizierungsmaßnahmen hervorrufen. Diese Abhängigkeiten sollten unter anderem aus Kostengründen vermieden werden. Es bedarf einer Übersicht aller Verbindungen und potentiellen Wirkketten innerhalb eines Softwaresystems.

Der SwcRank ist nicht auf die direkten Nachbarknoten einer sicherheitsrelevanten Komponente begrenzt. Er berücksichtigt die Wichtigkeit von Knoten in einer Kette vernetzter Komponenten. Somit können einzelne, hoch vernetzte Komponenten innerhalb eines Graphen identifiziert werden. Ein beispielsweise verursachter Fehler verbreitet sich somit in einer ähnlichen Art und Weise entlang der vernetzten Pfade. Das grundlegende Verhalten des PageRank Algorithmus wird für den SwcRank nicht verändert. Somit skaliert dieses Vorgehen auch zur Analyse größerer Softwarenetze (wie Google's Vorgehen zur Bewertung von Webseiten).

Für Mehrkernsysteme kann, neben dem Einfluss sicherheitsrelevanter Software, eine weitere Aussage zur Auffindung kommunikationsintensiver SWCs aus einem stark vernetzten Graphen abgeleitet werden. Existieren vereinzelte, hochrangige SWCs innerhalb des Netzwerkes, sind diese meist stark mit einer Vielzahl von weiteren Komponenten vernetzt. Sollen diese Komponenten auf unterschiedliche Rechenkerne verlagert werden, existiert unter Umständen kein zufriedenstellendes Verteilungsszenario (z.B. ausgewogene Rechenlastverteilung). Alle verbleibenden SWCs sind mit diesen Knotenpunkten verbunden. Somit verursachen hoch vernetzte SWCs ein hohes Kommunikationsaufkommen zwischen einzelnen Rechenkernen, egal auf welchem Kern diese

integriert sind. Eine Segregation bzw. Verteilung von stark vernetzten Komponenten ist in diesem Fall nicht sinnvoll.

Ist also die Wichtigkeit einer SWC im Vergleich zu ihren Nachbarn hoch, muss ein Kompromiss zwischen der Rechenlastverteilung von SWCs und der anfallenden Inter-Core-Kommunikationskosten getroffen werden. Zusätzlich müssen die Isolationsanforderungen zur Wahrung der Rückwirkungsfreiheit zwischen SWCs erfüllt werden. Je nach Anzahl der Abhängigkeiten zwischen Rechenkernen und Softwarepartitionen sollte daher das Softwarenetz und dessen Wichtigkeiten nach Möglichkeit homogenisiert werden, um einzelne Hotspots zu vermeiden. Ist dies nicht möglich, können SWCs aus einem Softwarenetz nicht herausgelöst werden. Andernfalls dürfen SWCs hinsichtlich der Isolationsanforderungen nicht gemeinsam innerhalb einer Partition integriert sein.

Manche Komponenten werden lediglich beim Aufstarten einer ECU aktiv. Zur Bewertung einzelner Knoten anhand ihrer Ein- und Ausgänge müssen daher zyklisch aufgerufene Signalpfade stärker gewichtet werden als sporadische Beziehungen, welche nur selten angestoßen werden. Dadurch werden schwach gewichtete Pfade während der Kalkulation des SwcRanks weniger berücksichtigt.

4.8 Zusammenfassung des Kapitels

Gemäß ISO 26262 [89] ist die Rückwirkungsfreiheit sicherheitsrelevanter Softwarekomponenten sicherzustellen. Die graphenbasierte Analyse von Softwarenetzen ist hierfür ein erster Ansatz, um eine Übersicht der Abhängigkeiten zwischen Fahrzeugfunktionen zu erhalten. Zum Zwecke der Ermittlung des Einflusses einzelner SWCs innerhalb eines Softwarenetzes, unter Berücksichtigung der Abhängigkeiten zwischen SWCs, wurde in dieser Arbeit ein Algorithmus zur Analyse sozialer Netzwerke entwickelt [120]. Dieser wurde an die Bedarfe der Automobilindustrie angeglichen. Es wurden etablierte Verfahren (der PageRank bzw. TrustRank) zur Analyse der Verlinkungsstruktur von Webseiten synthetisiert und an zusätzliche Eigenschaften sicherheitsrelevanter Softwarenetze angepasst.

Der SwcRank dient somit als erste Übersicht zur Architekturgestaltung und wurde eigens innerhalb dieser Arbeit entwickelt. Er wird verwendet um Abhängigkeiten zwischen SWCs ausfindig zu machen und die grundlegende Partitionierbarkeit eines Systems beurteilen zu können. Die grundsätzlichen Eigenschaften des PageRank Algorithmus wurden dabei nicht verändert. Somit eignet sich der SwcRank identisch zum PageRank auch für sehr große Graphen bzw. Softwarenetze.

Das erarbeitete Verfahren gibt in unterschiedlichen Netzhierarchieebenen Aufschluss über potentiell kommunikationsintensive Knotenpunkte und zeigt den Einfluss sicherheitsrelevanter Komponenten innerhalb eines Graphen auf. Dies wurde im Rahmen einer Fallstudie am Beispiel eines reellen Steuergerätes, unter Verwendung von SWCs mit einem hohen Vernetzungsgrad mit sicherheitsrelevanten Softwareanteilen, gezeigt.

Ist die Zentralität vereinzelter SWCs hoch, sollten jeweilige Abhängigkeiten bzw. Aufrufe benachbarter Komponenten verringert werden. Dies gilt auch für sicherheits-

relevante SWCs, falls diese innerhalb des Graphen hoch vernetzt sind. Durch eine mögliche Vermeidung dieser Abhängigkeiten soll unterbunden werden, dass ein festgelegtes ASIL einer sicherheitsrelevanten SWC auf nicht-qualifizierte SWCs vererbt wird (siehe Kapitel 3.3.3).

Die dargestellte Methode gibt somit Aufschluss über:

1. den Vernetzungsgrad eines klassischen und eines sicherheitsrelevanten Systems,
2. die Identifikation zentraler, kommunikationsintensiver SWCs, welche als Knotenpunkte im System agieren,
3. eine potentielle Prognose über die Querbeziehungen eines partitionierten Systems, sobald SWCs Softwarecontainern zugeteilt werden.

5 Ableitung von Kernelarchitekturen aus nicht-funktionalen Eigenschaften

Ziel dieses Kapitels ist die Entwicklung einer Methodik zur Ableitung einer Kernelarchitektur (KA) ausgehend von nicht-funktionalen Eigenschaften (NFEs). Daher wird eine Methode entwickelt, um anhand vordefinierter nicht-funktionaler Eigenschaften die geeignetste Software KA auswählen zu können. Bezogen auf die vorliegende Problemstellung innerhalb eines Softwareprojektes, resultiert aus dem folgenden Verfahren eine Entscheidungshilfe der optimalen KA für den Systemaufbau hochintegrierter Fahrzeugsoftwaresysteme. Mit der jeweiligen KA soll ein Großteil der zu erfüllenden nicht-funktionale Eigenschaften (NFEs) abgedeckt werden. Sie wird auf E/E-Softwaresysteme der Automobilindustrie angewandt. Der in dieser Arbeit verfolgte Ansatz zur hierarchischen Untergliederung von NFEs wird anhand von Baumstrukturen verwirklicht. Es wird grundsätzlich zwischen einer monolithischen, Mikrokern-basierten und einer Hypervisor-basierten KA unterschieden (siehe Kapitel 2.1 und 2.2).

5.1 Herausforderung und Problemstellung

Bereits Boehm et al. [39] und McCall et al. [106] beschreiben Factor-Criteria-Metrics (FCM) Modelle, um Softwarequalität übersichtlich und messbar innerhalb eines Modells auszudrücken [31]. Zusammenfassende Begriffe wie FURPS (*engl. Functionality, Usability, Reliability, Performance und Supportability*) werden von Grady and Caswell [63] definiert und beschreiben eine Zusammenstellung von Qualitätsanforderungen an Softwareprodukte. Standards wie ISO/IEC 25000 [90] bzw. ISO/IEC 9126 [91] fassen diese Ansätze zusammen und definieren bis heute Vorgehensmodelle zur Verbesserung der **Softwarequalität**.

Neben funktionalen Eigenschaften müssen eingebettete Softwaresysteme einer Vielzahl von **nicht-funktionalen Eigenschaften** gerecht werden. Hierzu zählen unter anderem Bedienbarkeit, Effizienz, Flexibilität, Integrität, Kompatibilität, Performanz, Portabilität, Robustheit, Sicherheit, Skalierbarkeit, Testbarkeit, Verfügbarkeit, Verlässlichkeit und Wartbarkeit [138]. Die zuvor erwähnten Vorgehensmodelle und Standards definieren Qualitätsattribute. Jedoch geben auch diese meist keinen Aufschluss, wie NFEs auf technischer Ebene umzusetzen sind.

Missionskritische bzw. geschäftskritische Systeme erfordern ein hohes Maß an **Zuverlässigkeit** (*engl. Dependability*, siehe Kapitel 5.2). Sie können während ihrer Lauf-

zeit nicht gewartet werden oder verursachen im Systemausfall hohe Kosten. Für eingebettete Fahrzeugsysteme mit meist langer Laufzeit ist die Zuverlässigkeit auch von erhöhter Wichtigkeit.

Innerhalb **sicherheitsrelevanter Systeme** müssen zusätzliche Mechanismen zur Einhaltung festgelegter Sicherheitsziele verlässlich und reproduzierbar greifen. Die Systemintegrität soll innerhalb dieser Systeme in jedem Fall gewährleistet sein, um Schaden für Leib und Leben abzuwenden. Aus Kundensicht ist Fehlerfreiheit bzw. -robustheit auch für Systeme ohne Sicherheitseinstufung relevant. Detaillierte Techniken und Methoden zur Erreichung dieser Eigenschaften sind in der ISO 26262 [89] weitgehend generisch ausgedrückt.

NFEs verfolgen meist **konträre Zielstellungen** und werden daher auf unterschiedlicher Art und Weise umgesetzt. Steigende Sicherheitsansprüche können unter Umständen die Effizienz eines Systems verschlechtern (z.B. Ressourcenvorhalt für Redundanzkonzepte). Die Erfüllung von NFEs stellt somit ein Optimierungsproblem für Softwareprojekte dar. Häufig sind dort nicht alle Ziele gleichermaßen bzw. vollständig erreicht.

Zur **Bestimmung der geeignetsten Kernelarchitektur** müssen NFEs in Relation gestellt werden. Es bedarf daher einer Methode, NFEs gegenüberzustellen und die geeignetste KA anhand ihrer Wichtigkeit für das individuelle Softwareprojekt zu identifizieren [50, 118].

5.2 Verwandte Methoden und Techniken

Der Begriff Zuverlässigkeit ist laut Laprie [101, 102] (siehe Abbildung 5.1) folgendermaßen untergliedert: Attribute (*engl. Attributes*) dienen als Indikator, um die Güte einer NFE zu definieren und berechenbar zu gestalten. Gefahren (*engl. Threats*) haben hierauf Einfluss und können die Güte dieser Eigenschaft mindern. Maßnahmen (*engl. Means*) sollen bekannte Bedrohungen eingrenzen und Fehlern entgegenwirken. Diese Überlegungen zur Aufteilung von NFEs können unter Umständen auf die Problemstellungen der Automobilindustrie übertragen werden.

Attribute

Zuverlässigkeit wird häufig durch das Akronym RAMS (*engl. Reliability, Availability, Maintainability und Safety*) ausgedrückt, welches innerhalb der DIN EN 50126 [55] definiert ist. Der Standard verweist unter anderem auf die genannten Qualitätsattribute, um Zuverlässigkeit weiter zu untergliedern. Diese werden in Kapitel 5.3 näher betrachtet. In der weiterführenden Literatur sind zusätzliche Attribute wie Integrität (*engl. Integrity*) und Vertraulichkeit (engl. *Confidentiality* in Verbindung mit *Security*) genannt [138]. Diese stehen jedoch nicht im Fokus dieser Arbeit.

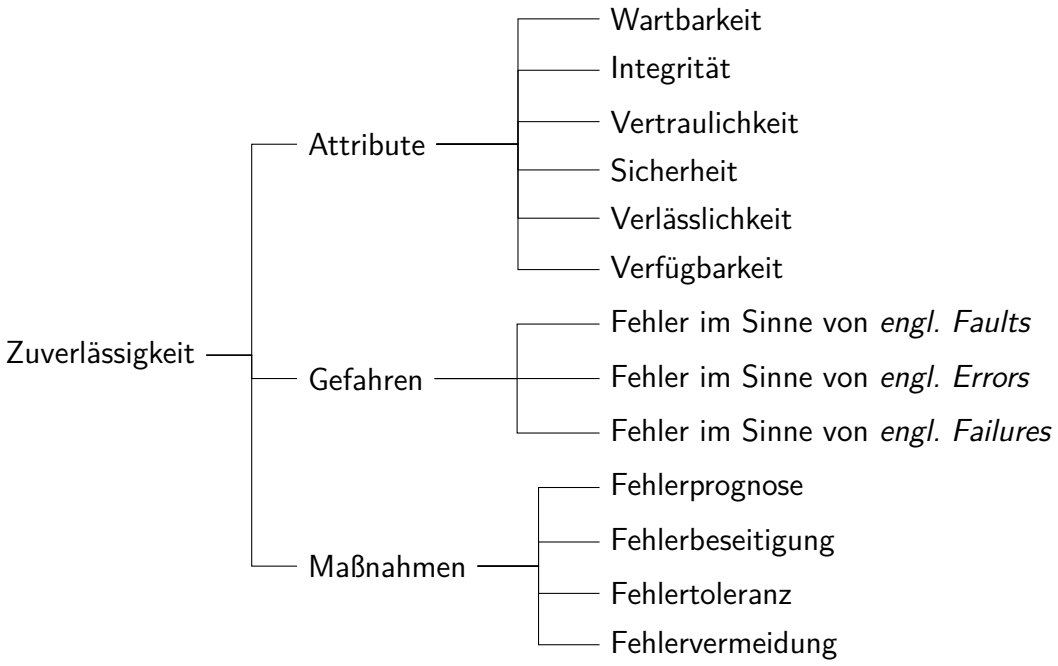


Abbildung 5.1: Aufschlüsselung der Systemzuverlässigkeit in Attribute, Gefahren und Maßnahmen nach Laprie [101].

Gefahren

Laprie [101] beschreibt das Systemversagen bzw. die Spezifikationsabweichung eines Dienstes als *Failure*. Diese Abweichung tritt aufgrund einer fehlerhaften Systemkomponente oder eines fehlerhaften Systemzustands auf. Der Teilbereich bzw. die Softwarefunktionalität des Systems, welche einen *Failure* verursacht, wird *Error* genannt. Die Fehlerursache für das Versagen einer Komponente oder eines Zustands bezeichnet Laprie [101] als *Fault*. Ein *Fault* kann aufgrund eines fehlerhaften Hardwarebauteils oder durch inkorrekten Programmcode entstehen.

Maßnahmen

Um die Zuverlässigkeit eines Systems zu steigern, müssen Maßnahmen und Methoden zur Fehlerbehandlung bereitgestellt werden. Hierfür klassifiziert Laprie [101] grundlegend nach aktiver und passiver Fehlerbehandlung.

Aktive Systemmaßnahmen und -funktionen (z.B. Softwareisolation) sind Maßnahmen, die bereits während der Systementwicklung berücksichtigt werden, um von Anfang an bestimmte *Fehlerbilder auszuschließen*. Treten Fehler zur Systemlaufzeit auf, werden Methoden verwendet, um bestimmte *Fehlerbilder tolerieren* zu können.

Unter **passiven** Methoden versteht man die Beseitigung von Entwicklungs- und Designfehlern durch intensive *Systemtests* während der Entwicklungszeit. Weiterhin sollen durch *Prognosewerkzeuge*, z.B. durch Fehlerbaumanalysen bzw. einer Fehlermöglichkeits- und -einflussanalyse (FMEA), Fehlerbilder eingegrenzt und deren Eintrittswahrscheinlichkeit gemindert werden.

5.3 Attribute der Systemzuverlässigkeit

Die zuvor in Kapitel 5.2 genannten Metriken der Systemzuverlässigkeit sollen nun erläutert werden. Wir orientieren uns am Vorgehen von Laprie [101] (siehe Abbildung 5.1) und verwenden die genannten Begrifflichkeiten der DIN EN 50126 (RAMS, siehe Kapitel 5.2).

5.3.1 Verlässlichkeit

Das Attribut Verlässlichkeit (*engl. Reliability*) bzw. Funktionsfähigkeit beschreibt die korrekte Ausführung eines Systems laut seiner Spezifikation innerhalb eines festgelegten Zeitraumes. Ein hohes Maß an Verlässlichkeit deutet darauf hin, dass das System seinem Nutzer weder falsche Informationen liefert, noch fehlerhafte Operationen durchführt oder vollständig ausfällt. Verlässlichkeit wird in der Literatur oft der Zuverlässigkeit gleichgestellt. Laprie [102] beschreibt Verlässlichkeit als Dienstbeständigkeit (*engl. continuity of service*).

> „*The reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.*“ [138]

> „*Reliability: a measure of the continuous service accomplishment (or, equivalently, of the time to failure) from a reference initial instant.*“ [101]

Auch Boehm et al. [38] definieren Softwarequalitätsattribute zur Unterteilung von NFEs. Softwareverlässlichkeit gliedert sich hiernach in *Autarkie, Vollständigkeit, Robustheit, Konsistenz und Genauigkeit* auf. Die aufgeführten Punkte beschäftigen sich mit der Entwicklung von Anwendungssoftware. Weitere Qualitätsmerkmale zur Verlässlichkeit von Software werden in der ISO/IEC 9126 [91] bzw. ISO/IEC 25000 [90] aufgegriffen. Diese unterscheiden zwischen *Produktreife, Fehlertoleranz und Wiederherstellbarkeit*. McCall et al. [106] untergliedern in Qualitätsfaktoren, -kriterien und -metriken. Dem Faktor Verlässlichkeit werden Einfachheit, Genauigkeit, Konsistenz und Fehlertoleranz als Kriterien unterstellt. Hierzu wird Verlässlichkeit in Produktreife und Robustheit untergliedert.

Die **Produktreife** beschäftigt sich mit der Beschaffenheit bzw. Qualität der Software und deren Design. Um verlässliche Ergebnisse produzieren zu können, sollen Kalkulations- und Arbeitsschritte standardisiert werden. Die Konsistenz einer Software beschreibt hierbei die Techniken und Richtlinien, mit welchen eine Software bzw. ein Steuergerät entwickelt wird. Es soll hierbei auf einen gemeinsamen Entwicklungsstandard (z.B. AUTOSAR) gesetzt werden. Festgelegte Schnittstellen sollen Fahrzeugfunktionen miteinander vernetzen. Einfachheit ist eine weitere Anforderung an die Softwareentwicklung. Hierbei sollen komplexe Strukturen (wenig Kohäsion/-Modularität) vermieden werden.

Die **Robustheit** einer Software soll deren Reaktion auf Fehler beschreiben. Somit sollen Softwarekomponenten eine gewisse Art und Anzahl von Fehlerbildern handhaben können. Neben der anhaltenden Verfügbarkeit des Gesamtsystems ist für ein

verlässliches System auch die Ausnahmebehandlung relevant. Eingabefehler dürfen somit nicht das Ausgabeergebnis verfälschen bzw. beeinträchtigen.

5.3.2 Verfügbarkeit

Die Verfügbarkeit (*engl. Availability*) des Systems definiert sich durch die Ausfallzeit des Systems im Verhältnis zu dessen gesamter Betriebsdauer. Um eine hohe Verfügbarkeit zu erlangen, muss das System jederzeit die spezifizierte Funktionalität ausführen können. Laprie [102] beschreibt Verfügbarkeit als *Nutzungsbereitschaft* (*engl. readiness for usage*).

> „*Availability: a measure of the service accomplishment with respect to the alternation of accomplishment and interruption.*“ [101]

> „*The availability of a system is the probability that it will be up and running and able to deliver useful services at any given time.*“ [138]

Durch Wartungsmaßnahmen, hervorgerufen durch Fehler bzw. Schäden, sind Systeme für die Zeitspanne ihrer Instandsetzung nicht verfügbar.

Die Systemverfügbarkeit drückt aus, mit welchen Unterbrechungen das System einen spezifizierten Dienst ausführen kann. Daraus folgt die **ununterbrochene Betriebszeit** (*engl. Uptime*) innerhalb eines festgelegten Zeitfensters. **Geplante Abschaltungen** können je nach Systemauslegung akzeptiert sein. Hierzu zählen Wartungsarbeiten, um aufgetretene Störungen zu beseitigen.

Je nach Funktionalität können Systeme im **Standby-Betrieb** verweilen bzw. zu festgelegten Zeiten eine **vollständige Abschaltung** veranlassen. Innerhalb eines Echtzeitsystems ist nicht nur die tatsächliche Betriebsfähigkeit aller vernetzter Partner notwendig, sondern auch die **Reaktions- und Antwortzeiten** müssen in einem festgelegten Zeitintervall erfolgen. Es wird zwischen **harten, weichen** und **festen Echtzeitanforderungen** unterschieden.

Werden harte Zeitschranken überschritten, ist das Systemverhalten fehlerhaft. Informationen bzw. Arbeitsschritte müssen innerhalb festgelegter Fristen erfolgen. Weiche Echtzeitanforderungen hingegen weisen keine strikten zeitlichen Vorgaben auf. Werden diese überschritten, hat dies kein fehlerhaftes Systemverhalten zufolge. Feste Echtzeitanforderungen definieren Zeitfenster, in denen ein Ereignis, wie ein Bearbeitungsschritt oder ein Signalempfang erfolgt sein muss. Ist dies nicht der Fall, entsteht kein Schaden. Das resultierende Ergebnis ist allerdings nicht weiter verwertbar.

5.3.3 Wartbarkeit

Die Wartbarkeit eines Systems ist eine breit gefächerte Eigenschaft. Unter Instandhaltbarkeit bzw. Wartbarkeit wird verstanden, ob ein System im Fehlerfall oder für Weiterentwicklungen nachträglich verändert werden kann (Updates). Die Wiederherstellungsfähigkeit eines Systems nach einem Ausfall ist ein weiteres Qualitätsattribut (QA) und ist eng mit bereits aufgeführten NFEs verbunden. Neben der Beseitigung

von Systemstörungen ist die nachträgliche Systemoptimierung bzw. Anpassbarkeit an neue Rahmenbedingungen ein wichtiger Faktor. Somit ist die Portabilität einer Softwarelösung maßgebend für einen generationsübergreifenden Einsatz innerhalb heterogener Systeme. Laprie [102] beschreibt Wartbarkeit als die Fähigkeit, sich Reparaturen und Weiterentwicklungen zu unterziehen (*engl. ability to undergo repairs and evolutions*).

> „*Software maintenance is the general process of changing a system after it has been delivered*“ [138]

Laut Sommerville [138] wird die Wartbarkeit von Software bzw. deren Änderungen folgendermaßen unterteilt: Die **Beseitigung von Fehlern** nach Auslieferung des Produkts ist vermutlich der bekannteste Änderungsgrund. Weiterhin müssen Softwaresysteme an **neue Betriebsumgebungen** angepasst werden. Für die Migration konsolidierter Systeme ist dies ein entscheidender Punkt. Innerhalb automotiver E/E-Systeme werden eine Vielzahl unterschiedlicher Betriebssysteme und Frameworks in unterschiedlichen Versionen betrieben. Da sich Produktanforderungen im Laufe des Produktlebenszyklus ändern, müssen häufig auch **Anpassungen** durch Updates und Erweiterungen vorgenommen werden.

Wartungsaufwand wird grundsätzlich mit Kosten in Verbindung gebracht und soll daher möglichst gering ausfallen. Ein beispielhaftes Maß zur Bewertung dieser Kosten ist die Anzahl der Änderungen und deren Bearbeitungsdauer. Je nach Komplexität einer Änderung ist der Aufwand höher oder geringer.

5.3.4 Funktionale Sicherheit

Funktionale Sicherheit (*engl. Functional Safety*) bedeutet die Vermeidung unzumutbarer Risiken, welche eine Gefahr für Menschen und deren Umwelt bedeuten. Um Sicherheit gewährleisten zu können, muss laut Sommerville [138] sichergestellt sein, dass Unfälle nicht entstehen können oder die Konsequenzen minimal sind. Laprie [102] beschreibt Sicherheit als das Nichteintreffen von katastrophalen Konsequenzen für die Systemumgebung (*engl. non-occurrence of catastrophic consequences on the environment*).

> „*Functional safety: absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems.*“ [89]

Die ISO 26262 [89] beschreibt funktionale Sicherheit ausschließlich im Rahmen von Kraftfahrzeugen zur Personenbeförderung. Sie behandeln dortige fehlerhafte Verhalten von sicherheitsrelevanten E/E-Systemen im Automobilsektor. Beim Systemdesign muss durch geeignete Maßnahmen das Sicherheitsziel (*engl. Safety Goal*) erreicht werden. Dabei muss festgelegt sein, wie bei Nichterfüllung eines Sicherheitsziels der sichere Systemzustand erreicht werden kann.

Neben dem primären Eigenschutz von Softwarekomponenten soll sekundär die Fehlerausbreitung anderer Komponenten und deren Einfluss auf sicherheitsrelevante Software erkannt und verhindert werden. Gemäß ISO 26262 [89] soll für sicherheitsrelevante Softwarekomponenten die Rückwirkungsfreiheit (*engl. Freedom from Interference*)

garantiert werden. Diese verlangt eine klare funktionale Trennung auf örtlicher (Speicher), zeitlicher (Ausführung) und informeller (Informationsaustausch) Ebene.

Primär dürfen Fahrzeugfunktionen und deren Softwarekomponenten nicht von anderen Softwareelementen gestört bzw. manipuliert werden. Dies ist Voraussetzung einer sicheren Systemausführung. Sicherheitsrelevante Softwarekomponenten müssen daher robust gegen abnormale Systemzustände sein. Dies bedingt die Implementierung swohl von Ausnahmebehandlungen und Rückfallkonzepten, als auch den direkten Schutz sicherheitsrelevanter Anwendungen vor nicht-vertrauenswürdiger Software.

Sekundär sind Schutzmaßnahmen zu treffen, um sicherheitsrelevante Anwendungen durch dritte Anwendungen zu schützen. Dies beinhaltet also einerseits die Erkennung von Fehlern und deren potentieller Fortpflanzung zwischen Komponenten (*engl. fault propagation*) und die Eingrenzung existierender Fehler und deren Kaskadierung (*engl. cascading failures*). Hierbei spricht man von einer Isolation nichtsicherheitsrelevanter Softwarekomponenten vom verbleibenden System.

5.4 Weitere nicht-funktionale Eigenschaften zur Systembewertung

Zur Identifizierung geeigneter KAs von Fahrzeugsoftwaresystemen müssen zusätzliche NFEs in Betracht gezogen werden.

5.4.1 Effizienz

Im Bereich der eingebetteten Systeme ist Echtzeitfähigkeit von hoher Wichtigkeit. Die Effizienz (*engl. Efficiency*) bzw. Performanz eines Systems ist ein entscheidender Faktor, da Hardwarekomponenten strikten monetären Rahmenbedingungen unterliegen. Eingebettete Systeme zeichnen sich durch den sparsamen Bedarf an Hardwareressourcen aus und sollen kostengünstig produziert werden.

> „*Effizienz ist die Fähigkeit des Softwareprodukts, ein angemessenes Leistungsniveau bezogen auf die eingesetzten Ressourcen unter festgelegten Bedingungen bereitzustellen.*“ [32]

Die Effizienz eines Systems untergliedert sich in dessen Zeit- und Ressourcenverbrauch. Im Fall eines eingebetteten Echtzeitsystems werden diese Attribute stark gewichtet, da diese dessen Grundcharakteristiken auszeichnen (kostengünstiger und planbarer Systemablauf).

Zeitliche Aspekte: IT-Systeme zeichnen sich durch einen möglichst performanten Ablauf aus. Die Antwort- bzw. Ausführungszeiten sollen möglichst kurz, der Durchsatz an verarbeiteten Daten gleichzeitig hoch ausfallen. Laut Boehm et al. [38] handelt es sich hierbei um Softwareeffizienz (*engl. Device Efficiency*) wie Zugriffsgeschwindigkeiten bzw. Verarbeitungszeiten und Erreichbarkeit bzw. Verfügbarkeit (*engl. Accessibility*).

„*Efficiency in time: the run-time performance of the program.*“ [106]

Für Echtzeitsysteme ist die verlässliche Vorausberechnung des Systemzeitverhaltens ausschlaggebend. Die Ausführung muss planbar sein, um oftmals harte Fristen einhalten zu können. In aktuellen Echtzeitbetriebssystemen der Automobilindustrie (wie OSEK/VDX) werden Tasks daher statisch allokiert. Diese sind somit in ihrer zeitlicher Ausführung und Reihenfolge gebunden. Gegensätzlich zu klassischen IT-Systemen, in denen die Taskausführung dynamisch verarbeitet wird, wird im Echtzeitbetrieb meist konservativ vorausgeplant.

Räumliche Aspekte: IT-Systeme sollen möglichst ressourceneffizient operieren und wenig Hardwareressourcen benötigen. Dies beinhaltet einerseits die Bereitstellung von ausreichend Programm- und Datenspeicher, andererseits auch das Vorhandensein von genügend Rechenkernen bzw. Hardwareschnittstellen, um eine sperrfreie Verarbeitung zu ermöglichen. Laut Boehm et al. [38] handelt es sich in erster Linie um quantifizierbare Ressourcen (*engl. Accountability*).

„*Efficiency in space: how much ressources are needed for operation.*“ [106]

5.4.2 Skalierbarkeit

In Kapitel 3.4.1 wurde der Skalierbarkeitsbegriff für diese Arbeit eingegrenzt. Zur weiteren Untergliederung in Qualitätsattribute und Metriken soll die NFE Skalierbarkeit näher erläutert werden. Hierfür werden die Vorgehensweisen von Neuman [111] und Sommerville [138] verwendet.

> „*Skalierbarkeit ist die Fähigkeit der Anpassung an wechselnde Lasten ohne Änderung grundsätzlicher Architekturen und Algorithmen.*“ [99]

Aus Sicht der Hardware erfolgt in einer **horizontalen Skalierung** eine Leistungssteigerung ausschließlich durch Hinzufügen von zusätzlichen Ressourcen (Beispiel: Mehrkernsysteme). In einer **vertikalen Skalierung** erfolgt eine Leistungssteigerung durch Verbesserung der einzelnen Ressourcen (Beispiel: Steigerung der Leistung einer CPU). Skalierbare Systeme sollen außerdem trotz Wegnahme von Ressourcen einen verhältnismäßigen Leistungsverlust aufweisen. Sommerville [138] hingegen spricht in seiner Definition ausschließlich von einer Hinzunahme von Ressourcen.

Konsolidierte, eingebettete Systeme bedingen striktere Skalierbarkeitsanforderungen. Hardwareressourcen sind eingeschränkt vorhanden und werden feingranular erhöht (z.B. zusätzliche Rechenkerne) bzw. aus Kostengründen rationalisiert (z.B. Erhöhung oder Verminderung der Taktrate). Diese Flexibilität sollen zukünftige E/E-Softwaresysteme aufweisen.

Aufgrund der auftretenden Verlustleistung, erläutert in Kapitel 1, kommt eine vertikale Skalierung für Steuergeräte in der Automobilindustrie nur sehr eingeschränkt in Frage. Die horizontale Skalierung von Rechenkernen ist daher vorzuziehen.

QAs werden jeweils in horizontaler und vertikaler Skalierung untergliedert. Werden Softwarekomponenten auf einer gemeinsamen Hardwareplattform hochintegriert, sollen die von Neuman [111] identifizierten Skalierungsdimensionen für verteilte Systeme verwendet und auf die Problemstellung adaptiert werden:

- **Administrationsaufwand**: Beherrschung der Systemkomplexität bei steigender Anzahl von unterschiedlichen Softwarekomponenten und -versionen.
- **Ressourcenbedarf**: Gemessener Bedarf (Skalierungsfaktor) an zusätzlichen Ressourcen (räumlich und zeitlich) bei Hinzunahme von zusätzlichen Fahrzeugfunktionen auf einem Steuergerät.
- **Verlagerbarkeit**: Gemessener Integrations- und Konfigurationsaufwand bei Hinzunahme bzw. steuergeräteübergreifenden Verlagerung von Softwarekomponenten.

Die deterministische Programmausführung beeinflussen soll nicht beeinflusst werden. Daher wird die aktive Beschleunigung integrierter Fahrzeuganwendungen auf eingebetteten Mehrkernsystemen durch Parallelisierung der Programmausführungspfade in dieser Arbeit nicht verfolgt. Stattdessen werden Fahrzeugsoftwarekomponenten vollständig auf individuelle Rechenkerne partitioniert. Um weiterhin zusätzlichen Rechenbedarf zu vermeiden, soll der Synchronisationsbedarf zwischen verlagerten Anwendungen minimalisiert werden. Werden zusätzliche Fahrzeugsoftwarekomponenten auf ein Echtzeitsystem integriert, sollen weiterhin alle Deadlines eingehalten werden können.

5.5 Methode zur Bestimmung geeigneter Kernelarchitekturen

Im Folgenden wird eine Methodik zur Bestimmung von KAs ausgehend von NFEs eingeführt. Das Vorgehen ist hierarchisch innerhalb einer Baumstruktur untergliedert. Die Methodik orientiert sich an der Herangehensweise von Laprie [101] zur Definition der NFE Zuverlässigkeit und untergliedert deren Qualitätsattribute.

5.5.1 Grundstruktur eines NFE-Baumes

Die Darstellungsweise von Laprie [101] soll für die Anforderungen hochintegrierter Systeme (siehe Kapitel 3) erweitert und konkretisiert werden (vergleiche hierzu Abbildung 5.4). Das Vorgehen zur Erstellung von NFE-Baumstrukturen für untergeordnete Knoten soll nun erläutert und in nachfolgenden Kapiteln vertieft werden.

Ziel des Vorgehens ist die Entscheidungsfindung der projektspezifisch geeignetsten KA, ausgehend von einer Menge von NFEs. Als Methodik sollen KAs von NFEs abgeleitet werden. Die NFE-Baumaufteilung ist hierarchisch und erfolgt in der in Abbildung 5.2 und 5.4 dargestellten Abfolge.

An der Spitze jedes Baumes steht das Projekt bzw. das spezifische Vorhaben, welches unter Umständen eine Vielzahl von NFEs bedingt. Hierbei leiten nichtfunktionale Eigenschaften (NFEs) die jeweiligen Entscheidungsbäume ein. Einer NFE werden Qualitätsattribute (QAs) untergeordnet, welche der weiteren Beschreibung

Projekt: spezifisches Vorhaben
↳ Nicht-funktionale Eigenschaften (NFEs): Beschreibung einer NFE
↳ Qualitätsattribute (QAs): Weitere Untergliederung einer NFE
↳ Metriken (Ms): Quantifizierung eines Attributes
↳ Fehlerpotentiale (FPs): Potentielle Gefährdungen
↳ Technische Maßnahmen (TMs): Präventionsmechanismen
↳ Kernelarchitekturen (KAs): Monolith, Mikrokern, Hypervisor

Abbildung 5.2: Hierarchische Ebenen des Entscheidungsbaumes.

einer Haupteigenschaft dienen. QAs können weiter untergliedert werden, bis jedes Attribut durch berechenbare Metriken (Ms) beschrieben werden kann.

Fehlerpotentiale (FPs) wirken einer Metrik entgegen, wodurch sich der Wert einer Metrik verschlechtert. Sie werden daher hierarchisch den Metriken untergeordnet und sind bis zu einer einzelnen Fehler- oder Gefahrenursache aufgeschlüsselt. Sie fungieren somit als Informationsquelle für Schwachstellen im System und liefern statistische Eingabewerte für vorgelagerte Metriken (z.B. Mean Time To Failure (MTTF) zur Ermittlung der durchschnittlichen Ausfallhäufigkeit).

Technische Maßnahmen (TMs) bzw. Lösungskonzepte geben Aufschluss um FPs entgegenzuwirken (siehe Abbildung 5.3) und den Wert einer Metrik verbessern. Sie nehmen Bezug auf die übergeordnete, potentielle Gefährdung einer Metrik bzw. des vorgelagerten Qualitätsattributes. Sie sind direkt den FPs untergliedert. An den Kind-Knoten aller TMs, bzw. an den Blättern des NFE-Baumes, werden KAs vernetzt. Diese realisieren die technische Umsetzung einer TM.

Qualitätsattribute und Metriken

Um NFEs in KAs überführen zu können, werden sie im ersten Schritt in QAs unterteilt. Metriken bilden hierbei die Grundlage zur Beurteilung eines QAs bzw. der übergeordneten NFE. QAs und deren Metriken kommen in einer Vielzahl von Bewertungsmethoden zur Klassifizierung von Software zum tragen. Eine auf Kennzahlen basierte Methode ist das COCOMO-Schätzverfahren zur Kostenabschätzung von IT-Projekten gemessen an den Code-Zeilen der Software [40, 149].

Das Aufstellen und Bewerten von Metriken stellt ein quantitatives Verfahren dar, indem qualitative Eigenschaften einer Software in vergleichbaren Zahlenwerten ausgedrückt werden. Die objektive Beurteilung von NFEs soll somit ermöglicht werden.

Fehlerpotentiale

Die Annahmen von Laprie [101, 102] zur Definition des Fehlertyps sind in Part 1 der ISO 26262 [89] übernommen. Es wird grundsätzlich zwischen *Fault*, *Error* und *Failure* unterschieden (siehe Abbildung 5.3). In den folgenden Baumstrukturen sollen diese Fehlerarten weiter aufgeschlüsselt und klassifiziert werden. Dabei werden tatsächliche Fehlerbilder auf technischer Ebene eingeordnet.

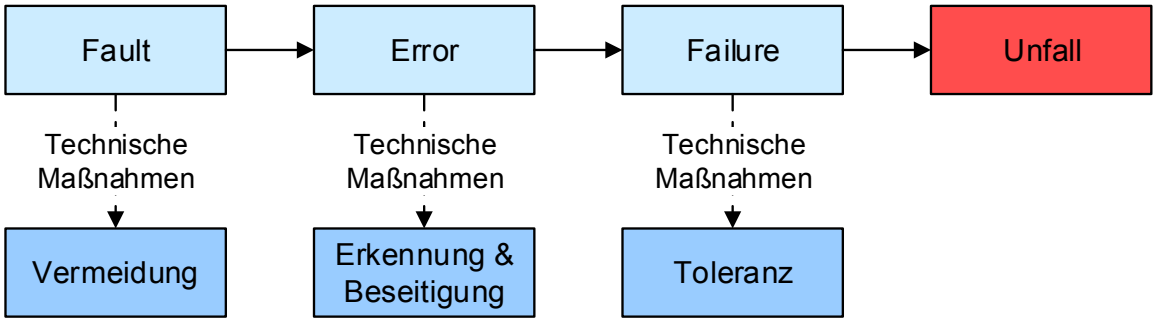


Abbildung 5.3: Fehlerwirkungskette im Zusammenhang von Fehlerpotentialen und technischen Maßnahmen.

Gemäß ISO 26262 [89] stellt ein *Fault* einen nicht-vorhergesehenen, abnormalen Zustand dar, welcher Einfluss auf das Systemverhalten haben kann. Die Wirkungskette in Abbildung 5.3 deckt sich mit den Aussagen von Laprie [101]. Der Übergang eines Systemfehlers kann unter Umständen zu einem tatsächlichen Unfall (*engl. Hazard*) führen und Personen gefährden. Daher sollen Übertritte zwischen einzelnen Fehlerklassen durch technische Maßnahmen (siehe folgender Abschnitt) unterbunden werden.

Technische Maßnahmen

Die in Abbildung 5.3 dargestellte Unterteilung orientiert sich an den Annahmen, genannt in [100–102]. Innerhalb dieser Arbeit liegt der Schwerpunkt auf den technischen Möglichkeiten zur Fehlervermeidung, speziell zur Systemlaufzeit.

Wie in Abbildung 5.3 dargestellt, untergliedern wir in drei grundsätzliche Maßnahmen, um eine potentielle Fehlerkette aufzubrechen:

- Fehlervermeidung (*engl. Avoidance*): Durch Entwicklungsrichtlinien und Softwarestandards wird während der Systementwicklung abgesichert, dass Programmfehler per Definition (siehe *Correctness-by-Construction* in [71, 81]) auszuschließen sind. Dies kann z.B. durch eine physische Trennung von Softwarekomponenten innerhalb eines Steuergerätes erreicht werden.

- Fehlererkennung und Beseitigung (*engl. Detection and Removal*): Mechanismen zur Detektion und Beseitigung von Fehlern (Hard- und Software) beugen Ausfällen zur Systemlaufzeit vor. Die meisten ECUs bieten hierfür mindestens Speicherschutzmechanismen (MPUs) an.

- Fehlertoleranz (*engl. Tolerance/Mitigation*): Nicht spezifizierte Systemzustände, Eingabewerte oder verursachte Fehler von dritten Komponenten können toleriert werden. Das System soll robust gegenüber einer bestimmten Art von Fehlerbildern sein. Es soll gewährleistet werden, dass das System seine spezifizierte Funktionalität u.U. auch im Fehlerfall garantiert ausführen bzw. selbst keine Gefahr für dessen Benutzer darstellen kann.

Kernelarchitekturen

Im Rahmen dieser Arbeit unterscheiden wir die KAs zwischen einem monolithischen, Mikrokern-basierten und Hypervisor-basierten Systemaufbaus (siehe Kapitel 2.1). Zur Konsolidierung von Fahrzeugsoftware auf einem Steuergerät (siehe Kapitel 3) soll dabei abhängig der geforderten NFEs die geeignetste KA gewählt werden. Dabei beschränkt sich die Betrachtungsauswahl auf monolithische Systemarchitekturen, den Einsatz eines Mikrokernels und die Verwendung eines HVs.

Die Blattknoten des Entscheidungsbaumes werden durch KAs dargestellt (siehe Abbildung 5.2). Somit enden alle Verzweigungen in einer Auswahl potentieller KAs zur Umsetzung einer TM. Diese wird durch eine KA in unterschiedlicher Güte umgesetzt. Als Resultat wird zwischen KAs projektspezifisch entschieden, um die angedachten NFEs des Zielsystems erfüllen zu können.

5.5.2 Graphenbasierte Darstellung und Gewichtung

Nicht-funktionale Eigenschaften und deren Anforderungen an ein System sind meist projektabhängig und gleichzeitig von unterschiedlicher Wichtigkeit. Um die Komplexität der vielfältigen Querbeziehungen zwischen einzelnen Eigenschaften bewältigen zu können, soll nun auf methodischer Art und Weise eine graphenbasierte Baumstruktur erstellt werden.

Exemplarisch soll nun ein Entscheidungsbaum einer NFE erstellt werden. Dabei werden die Kantenübergänge einzelner Knoten mit Kennzahlen bzw. Gewichten annotiert, wodurch projektspezifisches Wissen zur Lösungsfindung mit einfließen soll. Zusätzliche Gewichte sollen den jeweiligen Lösungswegen unterschiedliche Schwerpunkte verleihen (siehe Balance Scorecard [94]).

Hierzu wird gerichteter Graph G beschrieben, welcher durch das Tripel (V, E, W) ausgedrückt ist. Dabei bildet V die Menge aller Knoten $v \in V$, E die Menge aller Kanten $e \in E$ und W die Menge aller Gewichte $\omega \in W$. Jede Kante $e = (v1, v2) \in E$ wird durch die Teilmenge $E \subseteq V \times V$ abgebildet. Jede einzelne Kante drückt die gerichtete Beziehung zwischen Knoten, ausgehend von einer Quelle src(e) = v_1 verbunden mit einer Senke snk(e) = v_2 aus. Ein Quelle kann mehrere Senken referenzieren.

Alle Kanten $e \in E$ sind zusätzlich mit einem Kantengewicht $\omega \in W$ assoziiert. Die Kanten- oder Zweiggewichte, ausgehend von einem Elternknoten, sind normalisiert und deren Summe gleich 1 (siehe Abbildung 5.4, Ebene TM, Element fp_1: $\omega_{tm_1} + \omega_{tm_2} = 1$). Kantengewichte stellen die Relevanz des nachfolgenden Baumattributs hinsichtlich des Projektes dar. Sie basieren auf Erfahrungswerten oder experimentell ermittelten Ergebnissen. An dieser Stelle muss somit ingenieurmäßiges Wissen der jeweiligen Domäne projektspezifisch eingegeben werden. Dadurch kann das methodische Verfahren sowie dessen Entscheidungsresultate individuell gesteuert und dadurch beeinflusst werden.

Wir definieren die Mengen NFE = {nfe_1, nfe_2, ...}, QA = {qa_1, qa_2, ...}, M = {m_1, m_2, ...}, FP = {fp_1, fp_2, ...}, TM = {tm_1, tm_2, ...} und KA = {ka_1, ka_2, ...}, wobei diese Mengen wechselseitig disjunkt sind. Dabei werden die Knoten $v \in V$ werden gebildet durch V = {NFE $\cup$ QA $\cup$ M $\cup$ FP $\cup$ TM $\cup$ KA}. Jegliche Teilbäume

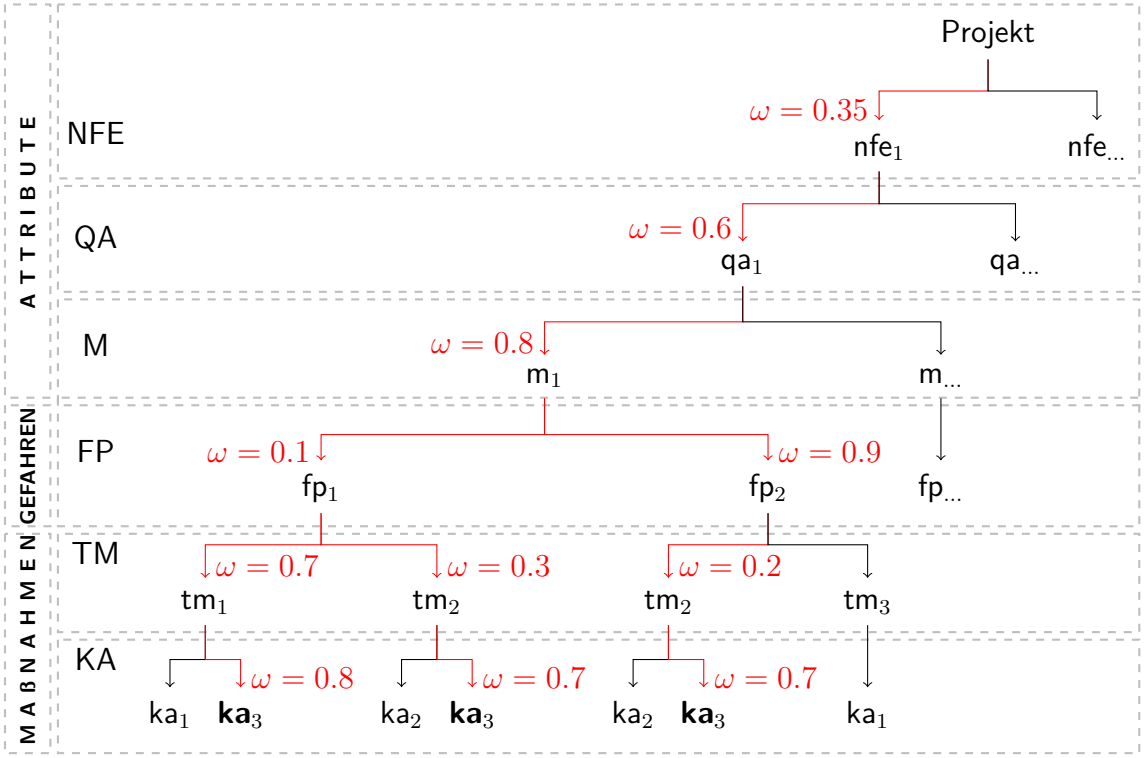


Abbildung 5.4: Exemplarischer Aufbau einer gewichteten Baumstruktur.

sind mit einer gemeinsamen Wurzel verbunden, welche das jeweilige Softwareprojekt symbolisieren soll.

G ist ein zusammenhängender, gerichteter und zyklenfreier Graph. Der Grad bzw. die Anzahl der ausgehenden Verbindungen eines Knotens wird als $deg^-(v)$ bezeichnet. Der Grad an eingehenden Verbindungen wird durch $deg^+(v)$ beschrieben. Graph G soll als Baum mit folgenden Eigenschaften dargestellt werden: Der Knoten $v \in V$ mit Grad $deg^+(v) = 0$ repräsentiert die Wurzel des Baumes. Alle Knoten $v \in V$ mit Grad $deg^-(v) = 0$ repräsentieren äußere Knoten bzw. die Blätter von G. Alle übrigen Knoten $v \in V$ bilden die inneren Knoten des Baumes G und besitzen $deg^+(v) = 1$ und $deg^-(v) \geq 1$.

Nach Erstellung des Baumes sollen alle Lösungspfade abgebildet sein, welche für die Erreichung einer bestimmten NFE und aus Sicht eines Projekts benötigt werden. Zur Identifikation der optimalen KA wird das Produkt der Kantengewichte aller Lösungspfade gebildet. Für jede KA wird die Summe der relevanten Pfadprodukte gebildet. Aufgrund der stochastischen Eigenschaft des Baumes ist die Summe aller Pfadprodukte gleich 1.

Zur Veranschaulichung wird in Abbildung 5.4 eine mögliche NFE-Baumstruktur gezeigt. Ziel ist die Ermittlung der optimalen KA. Beispielhaft soll das Gesamtgewicht $\omega_{ka_3,gesamt}$ berechnet werden. Hierzu wird entlang der rot markierten Pfade, welche ka_3 als Blattknoten aufweisen, das Produkt der Pfadgewichte ω gebildet und aufsummiert. Exemplarisch für das Gewicht von ka_3, ergibt sich die folgende Berechnung entlang aller gewichteten (ω) Zweigpfade: $0.35 * 0.6 * 0.8 * 0.1 * 0.7 * 0.8 + 0.35 * 0.6 * 0.8 * 0.1 *$

$0.3 * 0.7 + 0.35 * 0.6 * 0.8 * 0.9 * 0.2 * 0.7 = 0.009408 + 0.003528 + 0.021168 = 0.034104$. Da es sich um einen Entscheidungsbaum handelt, müssen die Gewichte $\omega_{ka_1,gesamt}$ und $\omega_{ka_2,gesamt}$ analog berechnet und miteinander verglichen. Dieser Schritt muss also für alle ka's wiederholt werden um eine finale Aussage zur Auswahl der optimalen KA treffen zu können.

5.5.3 Einführung von Querbeziehungen zur Reduktion der Komplexität

Zur Vermeidung von Redundanzen und einer gleichzeitigen Eindämmung von Fehlern sollen Entscheidungsbäume zusammengefasst werden. Hierfür werden Querbeziehungen sowohl innerer Baumknoten eines NFE-Baumes als auch zwischen NFE-Bäumen zugelassen. Vergleichbar zum in Kapitel 5.2 erläuterten Vorgehen von Laprie [101], werden NFE-Bäume in die Kategorien: Attribute (umfasst NFEs, Qualitätsattribute und Metriken), Gefahren (umfasst Fehlerpotentiale) und Methoden (umfasst Technische Maßnahmen und Kernelarchitekturen) unterteilt (siehe Abbildung 5.1 und 5.4).

Ausgehend von NFEs sind manche inneren Baumknoten identisch (siehe redundantes Element tm_2 aus Ebene TM in Abbildung 5.4). Die resultierenden Baumstrukturen werden somit groß und unübersichtlich. Ein Baum enthält redundante Elemente, wenn identische Teilbäume existieren. Entscheidungsbäume sollen deshalb an identischen Teilbäumen zusammengeführt und somit redundante Teilbäume wiederverwendet werden (Beispiel: fp_1 und fp_2 in Abbildung 5.4 verweisen beide auf tm_2). Voraussetzung hierfür ist, dass Teilbaumstrukturen eine sowohl identische Vernetzung als auch Zweigbewertung aufweisen.

Eine Teilbaum-Zusammenführung ist sowohl innerhalb eines Baumes, als auch zwischen mehreren NFE-Entscheidungsbäumen möglich (Beispiel: NFEs mit identischen QAs). Dieses Vorgehen sollte jedoch lediglich zur Erleichterung während der Modellierung des Entscheidungsbaumes erfolgen. Abbildung 5.5 zeigt eine exemplarische Konsolidierung innerhalb eines Baumes (siehe Abbildung 5.4). Dieser beinhaltet neben mehreren Wurzeln, den NFEs, auch Querbeziehungen zwischen den inneren Baumknoten. Aufgrund der hierarchisch gerichteten Struktur (siehe Abbildung 5.2) ist der Graph weiterhin zyklenfrei.

Durch die Verknüpfung von Teilbäumen entsteht ein Netz bzw. Graph, in welchem Kindknoten mehreren Elternknoten zugeordnet werden können. Als Konsequenz dieses Vorgehens verkleinert sich die Größe des Baumes. Allerdings geht die zuvor vorhandene Baumeigenschaft aufgrund von Querbeziehungen zwischen benachbarten Baumelementen verloren. Innerhalb der resultierenden Polyhierarchie können somit Diamant-Strukturen (*engl. diamond-problem*) auftreten. Diese verursachen unter Umständen Mehrdeutigkeiten während der Implementierung und erschweren dadurch die Erfassung jeglicher Teilbäume einer Baumstruktur. Zur finalen Ergebniskalkulation müssen somit Querbeziehungen (Diamantstrukturen) zwischen Knoten aufgelöst werden, um den Graphen in eine klassische Baumstruktur zurückzuführen.

Der in Abbildung 5.5 dargestellte Teilbaum, ausgehend von der Metrik m_1, würde folgendermaßen berechnet werden: Für ka_3 ergibt sich das Gewicht: $0.1 * 0.7 * 0.8 +$

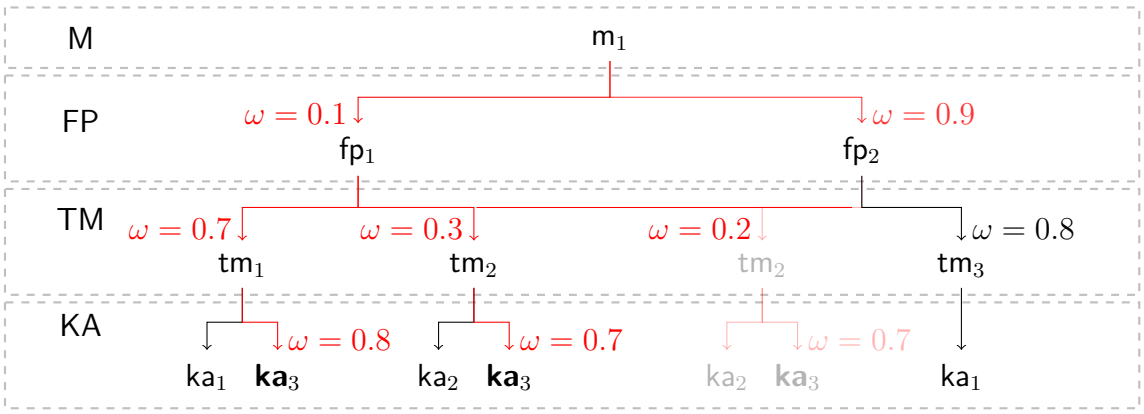


Abbildung 5.5: Zusammenführung redundanter Teilbäume.

$0.1 * 0.3 * 0.7 + 0.9 * 0.2 * 0.7 = 0.056 + 0.021 + 0.126 = 0.203$. Da der Teilbaum ab dem Element tm_2 doppelt im Baum vorhanden ist wird ersichtlich, dass innerhalb des Teilbaumes der Pfad $tm_2 \longrightarrow ka_3$ wiederverwendet wird.

5.5.4 Gegensätzliche Zielstellungen nicht-funktionaler Eigenschaften

Wie zuvor beschrieben, gehen die Grenzen nicht-funktionaler Eigenschaften ineinander über. Exemplarisch verlangt funktionale Sicherheit diverse Anforderungen, welche auch der Verlässlichkeit und Verfügbarkeit untergeordnet werden können. Qualitätsmerkmale mehrerer Eigenschaften bedingen bzw. ergänzen sich. Sie können jedoch auch komplementäre Ziele und Auswirkungen verfolgen. Bei der Aufstellung der Entscheidungsbäume, beschrieben in Kapitel 5.5.2, sind somit gegensätzliche NFEs bei der Vergabe von Gewichten zu berücksichtigen. Im Folgenden wird die Auswirkung von Eigenschaften aufeinander durch positive ($\oplus$) und negative ($\ominus$) Symbole gekennzeichnet.

Verlässlichkeit versus funktionale Sicherheit

Die Methoden und Systemanforderungen im Umgang mit Fehlern hinsichtlich Verlässlichkeit und funktionaler Sicherheit hängen stark zusammen. Laut Sommerville [138] ist funktionale Sicherheit dem Qualitätsmerkmal Verlässlichkeit untergliedert. Der Unterschied liegt in der Betrachtungsweise, wie sich ein Systemfehler auf Beteiligte auswirken könnte.

Systeme sollen grundsätzlich möglichst fehlerfrei operieren. Jedoch umfassen die Betrachtungen eines sicheren Betriebs ausschließlich Gefahrensituationen mit potentiellem Personenschaden (ISO 26262 [89] schützt in erster Linie Fahrzeuginsassen). Verlässlichkeit hingegen wird durch eine allgemeine Abweichung des Systemverhaltens von seiner Spezifikation innerhalb eines gewissen Zeitfensters ausdrückt (Metrik: MTTF).

Ein System kann grundsätzlich korrekt anhand seiner Spezifikation funktionieren (⊕ Verlässlichkeit). Jedoch kann eine Spezifikation fehlerhaft ausgelegt sein, wodurch die vermeintlich korrekt umgesetzte Funktion eine Gefahr für Leib und Leben darstellt (⊖ funktionale Sicherheit). Ein Beispiel hierfür ist das unkontrollierte Öffnen des Cabrio-Verdecks bei hohen Geschwindigkeiten.

Ein System kann weiterhin sicher sein (⊕ funktionale Sicherheit), jedoch durch restriktive Sicherheitsmaßnahmen das Gesamtsystem in seiner Ausführung behindern (⊖ Verlässlichkeit). Ein Beispiel hierfür ist das Erreichen des sicheren Zustandes bei transienten Fehlern durch Neustart des gesamten Systems. Dadurch entsteht kein Personenschaden, allerdings werden auch nicht betroffene Systemanteile zurückgesetzt. Dies wirkt sich negativ auf die Gesamtverlässlichkeit aus.

In den meisten Fällen gehen die Ziele der Systemverlässlichkeit und der funktionellen Absicherung eines Systems einher (z.B. elektrische Geräte mit Starkstromanschluss). Grundsätzlich sollen Systeme aus Qualitätsgründen für den Anwender verlässlich operieren (funktionale Sicherheit ist wichtiger als Verlässlichkeit). Allerdings sollte das wichtigste geforderte Systemziel die Vermeidung von Personenschaden sein.

Verlässlichkeit versus Verfügbarkeit

Die Verlässlichkeit eines Systems steht in direkter Verbindung mit dessen Verfügbarkeit. Wird ein bestimmter Dienst (z.B. Auslösen des Airbags) zu einem gewünschten Zeitpunkt nicht erbracht (⊖ Verfügbarkeit), so ist das System automatisch fehlerhaft und unzuverlässig (⊖ Verlässlichkeit). Gleichermaßen kann durch fehlerhafte Komponenten (⊖ Verlässlichkeit) der sichere Zustand durch Abschalten des Systems erreicht werden. Dadurch verringert sich wiederum die Verfügbarkeit (⊖ Verfügbarkeit).

Allerdings können beide Attribute bei identischem Systemverhalten konträre Ergebnisse aufweisen. Liefert ein System falsche Ergebnisse, hat dies negative Auswirkungen auf die Verlässlichkeit (⊖ Verlässlichkeit). Werden allerdings diese Anfragen zumindest zeitnah verarbeitet, wirkt dies positiv hinsichtlich der Verfügbarkeit (⊕ Verfügbarkeit).

Weiterhin sind in bestimmten Systemen und Situationen Attribute unterschiedlich gewichtet. Die Ansteuerung von Antriebssystemen (z.B. Motorsteuerung) erfordert meist ein Höchstmaß an Synchronizität und Timing. Allerdings sind fehlerhafte Eingaben unter Umständen tolerierbar (Verfügbarkeit ist wichtiger als Verlässlichkeit). Das Navigationssystem eines Fahrzeugs muss im Vergleich zwar verlässlich operieren, jedoch ist der tatsächliche Betrieb des Systems für den Fahrbereitschaft sekundär (Verlässlichkeit ist wichtiger als Verfügbarkeit). Die Gewichtung ist also abhängig vom betroffenen System und den angedachten Ausführungsszenarien.

Effizienz versus funktionale Sicherheit

Die Einhaltung von Sicherheitszielen und die damit verbundene Absicherung eines sicherheitsrelevanten Systems erfordert meist unabhängige Ausführungspfade und Rechenkerne (⊕ funktionale Sicherheit). Diese Redundanzen (räumliche Aspekte) ver-

schlechtern allerdings den Wiederverwendungsgrad innerhalb eines Systems und benötigen zusätzliche, exklusive Hardwareressourcen (⊖ Effizienz).

Werden Funktionen zur Signalplausibilisierung und Absicherung nicht verwendet, wird weniger Rechenkapazität (zeitliche Aspekte) in Anspruch genommen (⊕ Effizienz). Signale werden unter Umständen ungeprüft weitergeleitet bzw. verarbeitet. Dieses Vorgehen ist innerhalb sicherheitsrelevanter Systeme nicht tragbar und erschwert die Absicherung der betroffenen Funktionalität (⊖ funktionale Sicherheit).

Rückfallfunktionen und Notfallsysteme unterliegen häufig hohen zeitlichen Anforderungen und sollen im Fehlerfall schnell und zum korrekten Zeitpunkt greifen (siehe Airbag). Allerdings ist die Vermeidung von Personenschaden innerhalb sicherheitsrelevanter Systeme von höherer Wichtigkeit als die Erreichung von monetären Zielen, welche z.B. durch die Einsparung von Hardware erzielt werden könnten (funktionale Sicherheit ist wichtiger als Effizienz).

5.6 Ableitung Technischer Maßnahmen und Kernelarchitekturen am Beispiel funktionaler Sicherheit

Die in Kapitel 5.5 dargestellte Methodik wird nun anhand einer ausgewählten NFE evaluiert. Als Szenario dient hierfür die Konsolidierung von SWCs mehrerer Steuergeräte auf einer gemeinsamen ECU (siehe Kapitel 3.4.2). Dadurch soll die geeignetste KA für ein Softwareprojekt ausgehend von der ausgewählten NFE resultieren. Als Erwartungshaltung resultiert ein Architekturvorschlag, welcher aus einer monolithischen, Mikrokern-basierten bzw. Hypervisor-basierten Lösung ermittelt wird (siehe Kapitel 2.1).

Allerdings kann das dargestellte Vorgehensmodell nur schwer mit messbaren Ergebnissen verglichen werden, da Kantengewichte innerhalb eines Entscheidungsbaumes nicht direkt auf vergleichbare Größen zurückzuführen sind (Grundproblem einer nicht-funktionalen Eigenschaft). Daher hängt die Auswahl der Methodik stark vom jeweiligen Anwendungsfall ab und muss individuell erstellt und beurteilt werden. Um trotzdem die Güte der zuvor dargestellten Methode beurteilen zu können, werden folgende Evaluationsziele festgelegt:

- Abbildbarkeit auf vorherrschende Standards der Automobilbranche (wie AUTOSAR Administration [30], ISO 26262 [89], etc.).
- Anwendbarkeit auf realistische Szenarien und Problemstellungen innerhalb eines Fahrzeug-Softwareprojektes (siehe Kapitel 3.4.2: Hochintegrationsszenario zur Domänenleitrechner-Fusion).
- Dokumentationsmöglichkeit der situationsabhängigen Entscheidung für eine Kernelarchitektur innerhalb eines spezifischen Softwareprojektes.

Der Schwerpunkt dieser Arbeit liegt in der Hochintegration sicherheitsrelevanter E/E-Systeme. Daher wird im Folgenden die NFE funktionale Sicherheit fokussiert.

Diese ist in Kapitel 5.3.4 beschrieben und innerhalb einer Baumstruktur aufgeschlüsselt. Hierbei liegt der Fokus auf den TMs zur Behandlung potentieller Fehlerbilder, welche innerhalb der ISO 26262 aufgelistet sind. Während der Aufstellung einer Baumstruktur und der Gewichtung einzelner Knoten sollen die jeweiligen Kosten (Speicherverbrauch, Rechenbedarf, Integrationskosten, etc.) für TMs zur Priorisierung berücksichtigt werden.

5.6.1 Fehlerpotentiale

Die ISO 26262 [89] gibt diverse Fehlerbilder für sicherheitsrelevante Systeme vor, welche vor bzw. zur Laufzeit behandelt werden müssen.

Speicherfehler: Typischerweise wird in eingebetteten Systemen als Programmiersprache C/C++ verwendet. Es besteht dort die Möglichkeit, direkt auf Speicheradressen zuzugreifen. Durch fehlgeleitete Speicherzugriffe können Speicherzellen manipuliert oder gar korrumpiert werden.

Zeitliche Fehler: Durch Deadlocks, Livelocks oder andere blockierende Tasks können Anwendungen in ihrer Ausführung beeinträchtigt werden. Jegliche Art von fehlerhafter Allokation bzw. Reservierung von Rechenzeit kann zu Störungen im System führen. Somit entstehen zeitliche Interferenzen, die innerhalb der Spezifikation nicht berücksichtigt sind. Weiterhin ist die Synchronisation zwischen Softwarekomponenten und deren Ablaufreihenfolge zu berücksichtigen. Diese kann z.B. durch fehlerhafte Priorisierung von Tasks auftreten.

Fehler während des Informationsaustausches: Neben örtlichen und zeitlichen Fehlern können Fehlinterpretationen bzw. Manipulationen während des Informationsaustausches zwischen Softwarekomponenten auftreten. Eines der möglichen Fehlerbilder ist der Verlust von Nachrichten bzw. Informationen, welche zwischen Softwareeinheiten ausgetauscht werden. Weiterhin ist eine Manipulation der Daten innerhalb einer Nachricht während der Übertragung möglich. Im Echtzeitbetrieb müssen Nachrichten innerhalb eines festgelegten Zeitfensters eintreffen. Diese Fristen können wiederum aufgrund zeitlicher Fehler überschritten werden bzw. Nachrichten in falscher Reihenfolge beim Empfänger eintreffen.

5.6.2 Technische Maßnahmen

Um auf obige Fehlerbilder reagieren zu können, sind geeignete Sicherheitskonzepte notwendig. Exemplarisch werden Sicherheitskonzepte in Abbildung 5.6 innerhalb einer gewichteten Baumstruktur beschrieben und deren technische Umsetzung in Abbildung 5.8 konkretisiert. Wie in Kapitel 5.5 beschrieben unterteilen wir in:

Fehlervermeidung

Abbildung 5.7 führt die zuvor dargestellte Baumstruktur (siehe Abbildung 5.6) hinsichtlich Fehlermeidung fort. Bereits zur **Design- und Entwicklungsphase** können laut Vogel et al. [148] bekannte Fehlerbilder durch geeignete **Architekturmaßnahmen** softwareseitig ausgeschlossen werden. Um Interferenzen zwischen sicherheitsre-

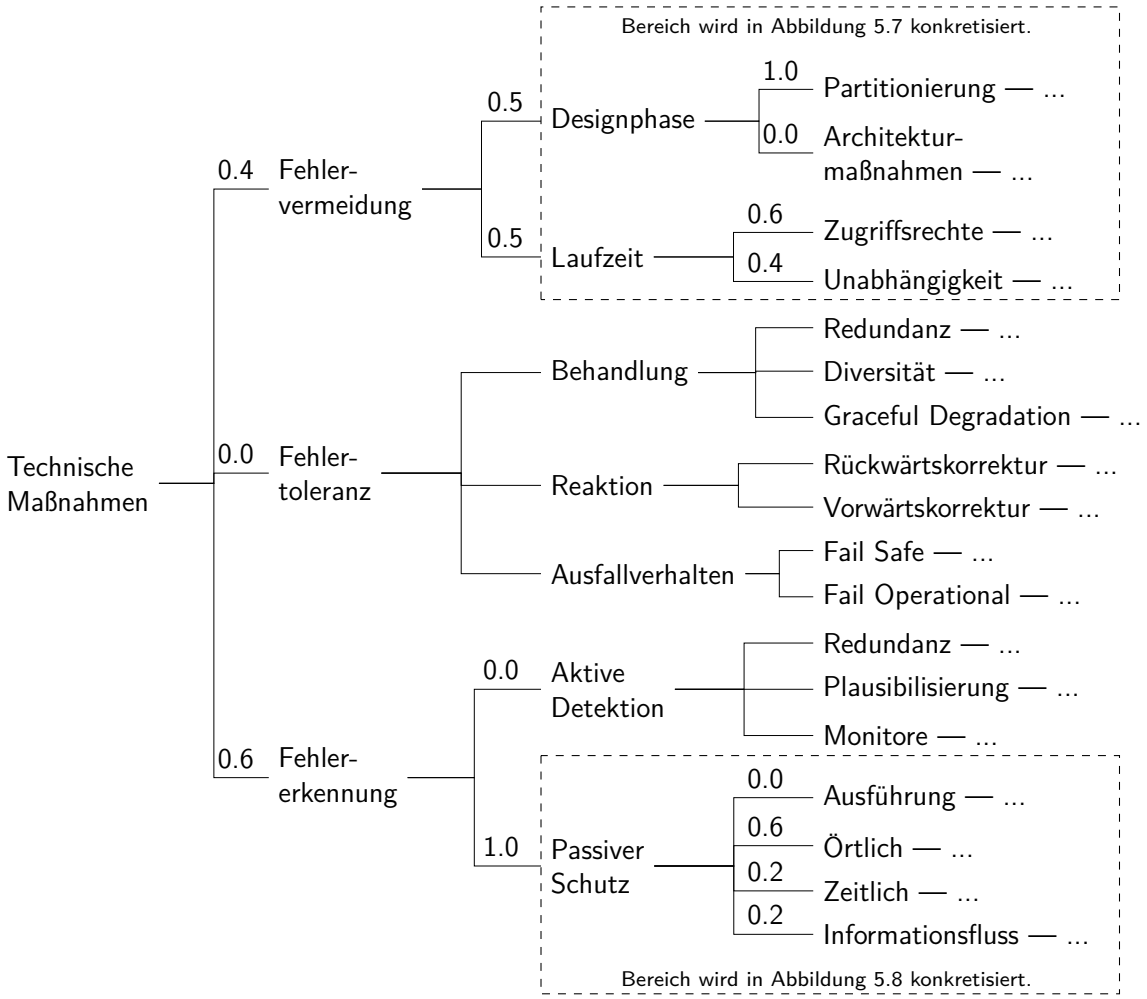


Abbildung 5.6: Maßnahmen zum Umgang mit Fehlern im Rahmen der funktionalen Sicherheit, dargestellt als Baumstruktur.

levanten und unkritischen Fahrzeugfunktionen zu vermeiden, sollen Softwarefunktionalitäten voneinander getrennt werden. In ISO 26262 [89] Band 6 werden typische Softwareengineering-Konzepte zur **Partitionierung** bzw. Kapselung von Software vorgeschlagen.

Sicherheitsrelevante Systeme sollen von nicht-qualifizierten Softwarekomponenten bereits innerhalb der System-**Designphase** abgekapselt werden [89]. Hierbei entsteht ein starker Zusammenhang zu den Forderungen einer geeigneten Softwarearchitektur, um eine feingranulare **Partitionierung** möglich zu machen [116]. Es wird hierbei zwischen einer Kapselung auf **Komponenten**- und **Systemebene** unterschieden. Die Granularität der Kapselung wird dadurch die jeweils eingesetzte Kernelarchitektur unterstützt.

Fehler können bereits durch geeignete Software-**Architekturmaßnahmen** verhindert werden [148]. Neben einer hohen Kohäsion innerhalb von Softwarekomponen-

ten muss die lose Koppelung bzw. **Modularität** zwischen Softwareeinheiten erfüllt sein [116]. Dies erfordert eine komponentenbasierte Entwicklung von Software, welche durch die Anwendung bekannter **Entwurfsmuster** nach Gamma et al. [61] unterstützt werden kann. Hierbei ist für eingebettete Systeme ein **objektorientierter Systementwurf** zur Organisation und Entwicklung großer Softwaresysteme hilfreich.

Zur **Laufzeit** können Fehlerbilder per Definition ausgeschlossen werden. Sicherheitsrelevante Fahrzeugsoftware benötigt aufgrund ihrer Qualitätsrichtlinien unterschiedliche Schutzkonzepte (wie Systemisolation). Qualifizierte Softwarekomponenten sind in erster Linie vom Fehlverhalten nicht-qualifizierter Komponenten zu schützen. Um unterschiedliche **Zugriffsrechte** auf Hardwareressourcen dediziert steuern zu können, ist ein Rechtemanagement erforderlich. Dies kann einerseits von der **Hardware** vorgegeben und durch CPU-Privilegierungsstufen eingeschränkt sein. Andererseits können Betriebssysteme, gestützt durch **Software**, den Zugriff von Prozessen auf Treiberschichten abstrahieren und einschränken. Daher bedingen diese Systeme unterschiedliche Privilegierungsstufen seitens der CPU (Superuser/User-Rechte), um den Zugriff auf Register und Hardwareressourcen, wie erweiterte Sicherheitsmechanismen (z.B. MPU), einzuschränken bzw. diese konfigurieren zu können.

Die **Unabhängigkeit** einzelner Softwarekomponenten untereinander ermöglicht den störungsfreien Betrieb. Sie wird von Softwaresystemen auf technischen Weg durch eine exklusive Zuweisung von Hardwareressourcen erreicht. Diese Exklusivität kann unter Verwendung einer Vielzahl von **Rechenkernen**, sowohl durch physikalisch getrennten **Speicher**, als auch individuellen **Peripheriegeräten** zur Kommunikation erreicht werden. Zur Vermeidung zeitlicher Interferenzen können Mehrkernsysteme somit echte Parallelausführung bereitstellen.

Fehlertoleranz

Die Beherrschung erkannter Fehler ist essentiell zur Gewährleistung eines rückwirkungsfreien Betriebs. Ein Fehlerfall kann einerseits korrekt behandelt und andererseits vom System in seiner Ausführung toleriert werden. Ist ein System- oder Komponentenfehler unvermeidbar, darf dieser zu keinem Personenschaden führen.

Zur weiteren **Behandlung** bestimmter Fehlerbilder müssen geeignete Maßnahmen vorab im System implementiert werden. Ein verlässliches System sollte gegen unbekannte Fehlerbilder robust sein bzw. diese durch geeignete Konzepte tolerieren. Dies kann durch **Redundanz** von Softwarekomponenten ermöglicht werden. Im Fehlerfall einer Komponente übernimmt eine weitere Einheit den vollwertigen Betrieb. Eine Weiterführung dieser Methode erfolgt durch die Einführung von **Diversität**. Wird die spezifizierte Funktionalität von Softwarekomponenten unterschiedlich implementiert, übernimmt im Fehlerfall die benachbarte Komponente den vollwertigen Betrieb (z.B. Steer-by-Wire vs. mechanische Lenkung als Rückfallebene). Um bei Fehlerverhalten einzelner Dienste den relevanten Betrieb aufrecht zu erhalten, wird die Systemausführung auf wesentliche Kernkomponenten reduziert. Dabei erfolgt eine **Degradierung** von niederprioren Systemkomponenten (*engl. graceful degradation*), um hochpriore Aufgaben weiterhin durchführen zu können (z.B. für sicherheitsrelevante Funktionen).

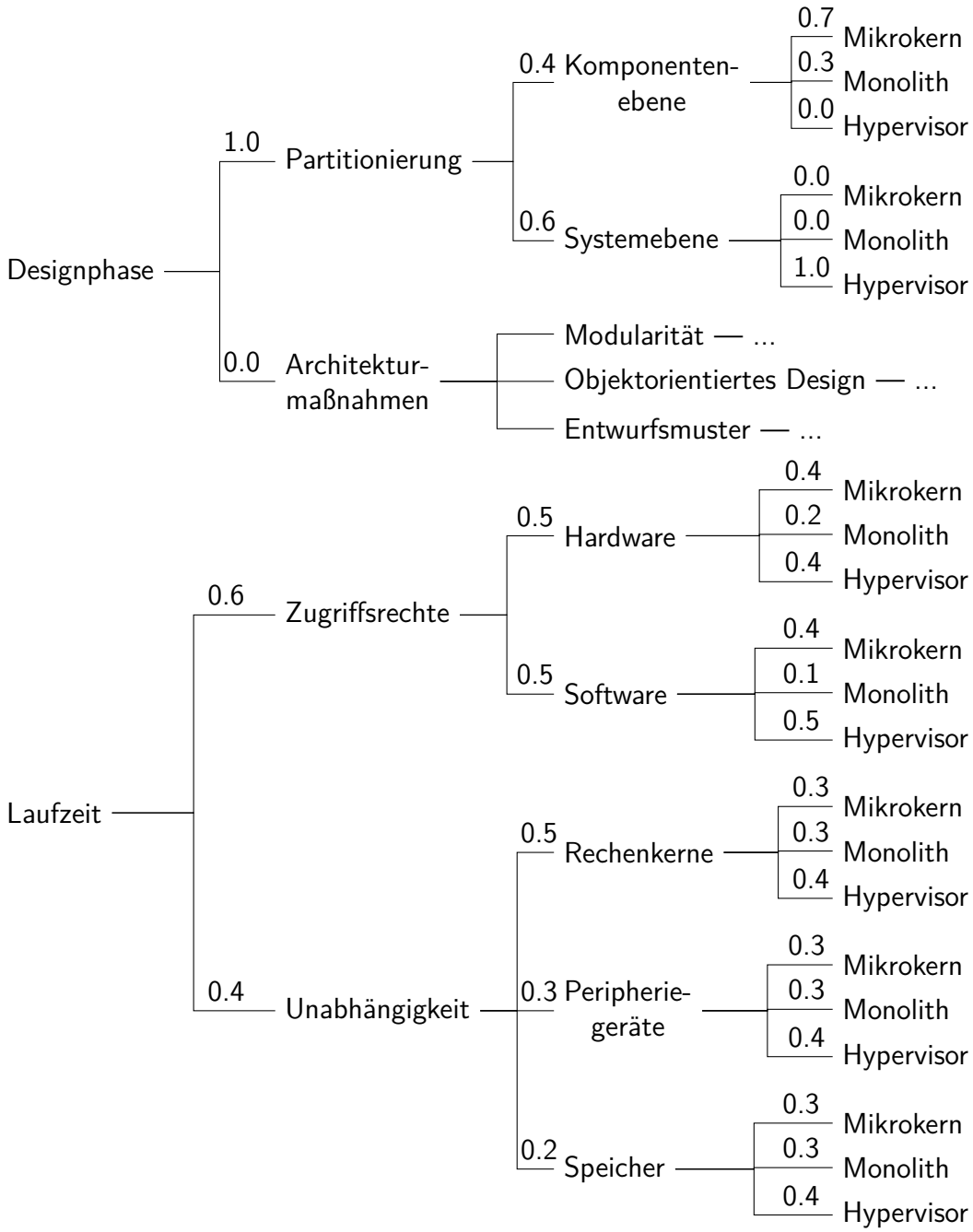


Abbildung 5.7: Detailausführung zur Fehlervermeidung von Abbildung 5.6.

Dabei bewirkt ein erkannter Fehlerfall eine **Reaktion** (Ausnahmeroutine) innerhalb des Systems. Werden keine weiteren Korrekturmaßnahmen eingeleitet, folgt die **Fortsetzung** des standardmäßigen Systembetriebs (*engl. forward recovery*). Ist der Fehlerfall schwerwiegender, erfolgt eine **Zurücksetzung** des Systems auf den zuletzt bekannten fehlerfreien Zustand (*engl. backward recovery*). Dieser Vorgang wird in der Datenbanktechnik auch als *Rollback* bezeichnet.

Zur Auslegung eines fehlertoleranten Systems muss das **Ausfallverhalten** im Fehlerfall vorab spezifiziert sein. Ein System gilt als **Fail Safe**, sobald es im Fehlerfall lediglich in einen sicheren Zustand versetzt werden muss, um keine ernsthafteren Schäden zu verursachen. Ein System gilt als **Fail Operational**, wenn es seine korrekte Ausführung trotz eines Fehlers weiter fortsetzen muss, um keine Gefährdung der Fahrzeuginsassen hervorzurufen.

Fehlererkennung

Die Erkennung und Reaktion auf transiente bzw. sporadische Fehler erfolgt hierbei zur Laufzeit. Eine Reaktion äußert sich entweder in der direkten Beseitigung eines auftretenden Fehlers oder in die Überführung des Systems in einen sicheren Zustand. Ein Fault kann zu einem Error führen, welcher sich wiederum zu einem System-Failure ausprägen kann (siehe Abbildung 5.3). Um diese Kette zu unterbrechen, müssen Maßnahmen implementiert sein, welche meist unvorhergesehene Fehlerbilder (wie transiente Fehler) beseitigen. Dazu müssen Fehler zuerst erkannt und nachfolgend behandelt werden. Wir untergliedern hierzu in aktive Detektion von Fehlern (mit Intelligenz) und passiven Schutz (ohne Intelligenz).

Durch die **aktive Detektion** von Fehlern werden potentiell transiente Fehlerfälle aufgedeckt. Um Fehlinterpretationen (z.B. false positives/negatives) entgegenzuwirken, sollen Informationen zur Laufzeit überprüft werden. Hierzu werden Interaktionen und Abläufe zwischen Softwarekomponenten (z.B. durch **Plausibilisierung**) überwacht, um die Informationsqualität bzw. Wertebereiche zu prüfen. Informationen werden somit auf deren Validität im Programmablauf bzw. Quellcode selbst abgesichert.

Softwarebasierte **Monitore** stellen hingegen intelligente Signal- und Informationsqualifizierung zur Verfügung und ermöglichen zusätzliche Redundanzkonzepte. Besonders bei zeitkritischen Abläufen ist die Überwachung des Programmablaufs notwendig, um Reihenfolgeproblemen entgegenzuwirken und die Programmausführungszeit nicht zu beeinträchtigen.

Neben der direkten Überwachung von Softwarekomponenten stellt diversitäre Programmierung durch **redundante Softwarekomponenten** (*engl. independant parallel redundancy*) eine weitere Möglichkeit zur Erkennung und Beseitigung von Fehlern dar. Basierend auf unterschiedlichen Eingabeparametern wird somit durch Erzeugung von Redundanzen der Wahrheitsgehalt von Ausgabewerten gegengeprüft.

Um eine mögliche Kaskadierung bzw. Vererbung von Fehlern vorzubeugen, müssen Softwarekomponenten voneinander geschützt werden. **Passive Mechanismen** (meist gestützt durch Hardwaremechanismen) reagieren auf einen fehlerhaften Zustand und lösen meist eine Ausnahmeroutine aus. Grundsätzlich sind Interferenzen sicherheitsrelevanter Softwarekomponenten jeglicher Art zu vermeiden. Daher werden z.B. Zugriffsrechte nicht-qualifizierter Anwendungen auf HW-Ressourcen (z.B. CPU-Register) eingeschränkt. Jedoch können nicht-qualifizierte Anwendungen trotz eingeschränkter Rechte zu Interferenzen führen. Die ISO 26262 [89] unterscheidet hierbei zwischen Fehlern in Ort, Zeit und während des Informationsaustauschs. Zur Fehlerde-

tektion - verursacht durch nicht-qualifizierte Softwarekomponenten - werden folgende passive Schutzkonzepte (siehe Abbildung 5.8) adressiert:

- **Örtlicher** Schutz: Eine MPU bzw. Memory Management Unit (MMU) schränkt den Zugriff auf Speicherbereiche ein (wie Register oder andere Softwareeinheiten). Eine abgeschottete Anwendung kann nur Speicherzellen in ihren dazu vorgesehenen Speicherbereich verändern. Wird fälschlicherweise auf anwendungsfremde Sektionen zugegriffen, wird eine Ausnahmebehandlung gestartet. Die geschützte Speicherstelle kann nicht manipuliert werden.

- **Zeitlicher** Schutz: Um die zeitliche Ausführung sicherheitsrelevanter Softwarekomponenten zu garantieren und zeitlichem Fehlverhalten entgegenzuwirken, müssen Deadlocks bzw. Laufzeitverletzungen frühzeitig erkannt werden. Hierfür können Watchdogs eingesetzt werden. Können diese Timer aufgrund eines Deadlocks oder zu langer Programmausführung anderer nicht zurückgesetzt werden, wird eine definierte Ausnahmebehandlung ausgeführt. Identisch hierzu existieren weitere hard- und software-basierte Timer verwendet, um mögliche Zeitverzüge zu festzustellen und im Fehlerfall Ausnahmeroutinen zu aktivieren.

- Schutz des **Informationsaustauschs**: Der Informationsfluss zwischen sicherheitsrelevanten Softwarekomponenten soll abgesichert werden. Um einer möglichen Signalmanipulation entgegenzuwirken, werden z.B. Prüfsummen eingeführt. Weiterhin wird auch die zeitliche Abfolge kontrolliert, ob Nachrichten zu früh oder verspätet beim Empfänger eintreffen. Weiterhin können zur Informationsplausibilisierung redundante Signalwerte miteinander verglichen werden. Somit können falsche Ergebnisse verglichen mit einer Vielzahl korrekter Signalwerte ermittelt werden.

- Schutz zur **Ausführung**: Zusätzlich zu den genannten Schutzkonzepten sollen Applikationen in deren Ausführung (durchlaufener Code) beschränkt sein. Dies ermöglicht die Umsetzung erweiterter Schutzkonzepte hinsichtlich Informationssicherheit. Der Ausführungsschutz kann unter anderem durch die Adressregionen innerhalb einer MPU abgesichert werden.

5.6.3 Auswertung der gewichteten Ergebnisse

Die exemplarische Vorgehensweise zur Evaluation der Entscheidungsmethodik anhand der NFE funktionale Sicherheit soll nun anhand der vergebenen Zweiggewichte ausgewertet und mit den vorgegebenen Evaluationszielen abgeglichen werden.

Auswertung zur Ermittlung der optimalen Kernelarchitektur

Zur Migration bestehender Softwaresysteme soll der zeitliche und monetäre Aufwand minimal sein. Im dargestellten Hochintegrationsszenario wird die Fehlererkennung stärker priorisiert, da diese meist lediglich auf einer Neukonfiguration der Ansteuerung

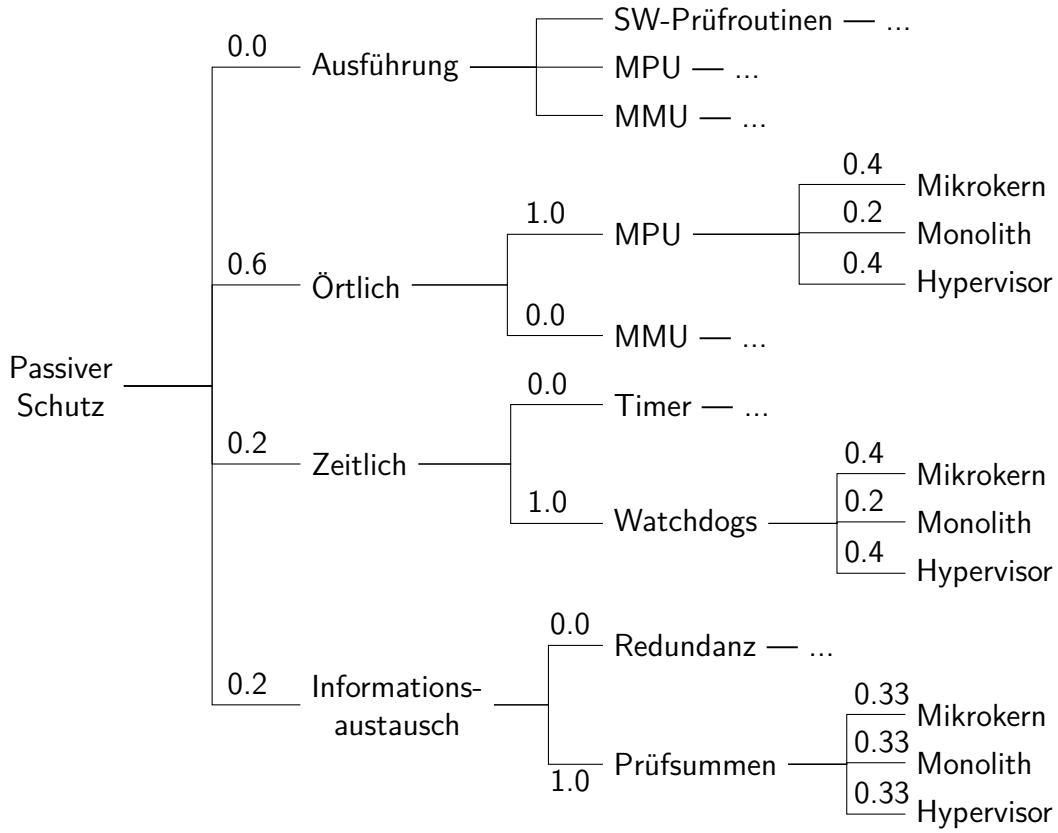


Abbildung 5.8: Detailausführung der Fehlererkennung zur Designphase und Laufzeit (siehe Abbildung 5.6).

hardwaregestützter Schutzmechanismen basiert. Dabei sollen in erster Linie Schutzmaßnahmen konfigurationsbedingt umgesetzt werden und keine weiteren Sicherheitsmechanismen implementiert werden.

Abbildung 5.6 stellt eine grundsätzliche Aufstellung dar, um Fehlerfällen entgegenzuwirken. Fehlertolerante Systeme ($\omega = 0.0$) im Rahmen des Hochintegrationsszenarios (siehe Kapitel 3.4.2) werden in diesem Beispiel ausgeschlossen. Hierfür verwendete Softwaresysteme verweilen im Fehlerfall lediglich in einem sicheren Systemzustand und lassen keine weitere Ausführung zu.

Fehlervermeidende Sicherheitsvorkehrungen ($\omega = 0.4$) sind meist leichter durch eine Neuimplementierung des Softwaresystems umzusetzen, verursachen allerdings höhere monetäre und zeitliche Aufwände. Sie werden durch Abbildung 5.7 weiter vertieft. Fehlervermeidung kann sowohl zur Laufzeit ($\omega = 0.5$) als auch zur Designphase ($\omega = 0.5$) implementiert sein. Deren Zweigbewertungen sind im aktuellen Beispiel gleichermaßen bewertet.

Zur Designphase sollen Architekturmaßnahmen ($\omega = 0.0$) zur Softwarekonsolidierung ausgeschlossen werden, da zu migrierende Systeme bereits existieren und nicht merklich erweitert werden sollen. Daher liegt der Schwerpunkt auf der Partitionie-

rung ($\omega = 1.0$) bzw. Verteilung der zu konsolidierenden Softwarekomponenten auf der neuen Zielplattform.

Die Zusammenführung auf Komponentenebene ($\omega = 0.4$) erfordert zusätzliche Konfigurationsaufwände innerhalb der AUTOSAR BSW. Aus Sicht der funktionalen Sicherheit bietet ein Mikrokern ($\omega = 0.7$) bessere Separationsmechanismen zur Trennung von Zugriffsrechten einzelner Softwareschichten als ein monolithisches System ($\omega = 0.3$), was die stärkere Priorisierung rechtfertigt. Ein HV ($\omega = 0.0$) für eingebettete Systeme wird im Rahmen eines Hochintegrationsszenarios lediglich zur Kapselung von VMs verwendet und bietet keine feingranularere Partitionierungsmöglichkeiten an.

Da innerhalb dieser Arbeit Fahrzeugfunktionen zusammengefasst werden, liegt ein stärkerer Fokus in der Partitionierung auf Systemebene ($\omega = 0.6$). Die Zusammenführung von Softwaresystemen wird lediglich durch einen HV ($\omega = 1.0$) ermöglicht und bisher nicht von typischen AUTOSAR Betriebssystemen unterstützt.

Zur Laufzeit ($\omega = 0.5$) wird die Verwaltung der Zugriffsrechte ($\omega = 0.6$) auf Hardwareressourcen und deren Umsetzung in Software vorerst gleichwertig gewichtet. Anders als nicht-qualifizierte Softwarekomponenten besitzen sicherheitsrelevante Funktionen meist Vollzugriff auf bereitgestellte Hardwareressourcen ($\omega = 0.5$), da von diesen Komponenten ein funktional abgesicherter Betrieb vorausgesetzt wird. Die Verwaltung der CPU-Priveligierungsstufen durch einen Hypervisor ($\omega = 0.4$) wird identisch zu einem Mikrokern ($\omega = 0.4$) priorisiert. Beide Varianten weisen meistens eine ähnlich kleine TCB auf. Somit besitzen nur sehr kleine Softwareteile im System tatsächlichen Vollzugriff und sind auf die Freigabe durch den Hypervisor oder durch den Mikrokern angewiesen. Innerhalb eines monolithischen Systems ($\omega = 0.2$) besitzen größere Anteile (z.B. Treiberschichten) Vollzugriff auf Hardwareressourcen und Register. Dadurch ist die Kontrolle bzw. Einschränkung der jeweiligen Zugriffe komplizierter bzw. die Absicherung der Softwareteile mit Vollzugriff aufwendiger.

Wird die Kapselung und Freigabe von Ressourcen im Softwaresystem ($\omega = 0.5$) an sich betrachtet unterscheiden sich Hypervisor und Mikrokern kaum. Allerdings ist ein Hypervisor ($\omega = 0.5$) in einer tieferliegenden Softwareschicht angesiedelt und erhält somit eine etwas stärkere Gewichtung als ein Mikrokern ($\omega = 0.4$), hinsichtlich der strikten Trennung und Ausführung von Kernel-Rechten und User-Rechten. Innerhalb eines monolithischen Systems ($\omega = 0.1$) werden weiterhin große Teile im Kernel bzw. Kernel-Modus betrieben und kanalisieren die Freigaben auf Hardwareressourcen in keiner Weise im Vergleich zu Mikrokern- oder Hypervisor-basierten Systemen.

Um Laufzeitproblemen vorzubeugen, können Hardwareressourcen unabhängig ($\omega = 0.4$) bzw. exklusiv einzelnen Softwaresystemen zur Verfügung gestellt werden. Dies ist allerdings nicht immer möglich, da besonders in eingebetteten Systemen bestimmte Hardwareelemente in begrenzter Anzahl zur Verfügung stehen. Da mehrere kleinere Systeme konsolidiert werden sollen, muss in erster Linie ausreichend Rechenkapazität ($\omega = 0.5$) bereitgestellt werden. Weiterhin müssen Kommunikations- und Peripheriecontroller ($\omega = 0.3$) bestenfalls exklusiv zugewiesen werden. Ist dies aufgrund fehlender Hardwareelemente nicht möglich, müssen Signale über einen Controller kanalisiert werden. Zuletzt soll der Programmspeicher ($\omega = 0.2$) zwischen

Applikationen bzw. Systemen aufgeteilt werden. Hierfür muss speziell beachtet werden, dass Zugriffe auf einzelne Speicherbänke während ihrer Ausführung nicht andere Fahrzeugfunktionen blockieren und somit deren Ausführung beeinträchtigen.

Für die aufgeführten Zweigabschnitte zur unabhängigen Zuweisung von Hardwareressourcen ist davon auszugehen, dass die Verwaltung bzw. Aufteilung einzelner Hardwareressourcen von monolithischen ($\omega = 0.3$) und Mikrokern-basierten ($\omega = 0.3$) Systemen identisch durchgeführt werden kann. Diese befinden sich innerhalb der gleichen Softwareschicht und abstrahieren die Konfiguration der Hardware auf ähnliche Art und Weise. Ein Hypervisor ($\omega = 0.4$) ist in einer tieferen Softwareschicht unterhalb der AUTOSAR BSW angebracht und ist in erster Linie für die grobe Aufteilung von Systemressourcen verantwortlich. Somit existieren hierbei mehr Freiheitsgrade zur Abstrahierung integrierter Systeme.

Eine Fehlererkennung ($\omega = 0.6$) erfolgt sowohl durch aktive bzw. intelligente Systemkomponenten, als auch durch passive bzw. statisch vorkonfigurierte Schutzmechanismen. Die Aufwände zur Konsolidierung mehrerer Softwaresysteme sollen gering gehalten werden. Daher sollen keine weiteren aktiven Detektionsmechanismen ($\omega = 0.0$) implementiert werden. Passive Schutzmechanismen ($\omega = 1.0$) werden meist von der Hardware zur Verfügung gestellt und erfordern lediglich eine korrekte Konfiguration und Verwendung innerhalb des Softwaresystems.

Abbildung 5.8 gliedert die Umsetzung des Speicherschutzes in drei potentielle KAs. Insbesondere für sicherheitsrelevante Systeme muss die korrekte Konfiguration und Steuerung gewährleistet sein. Daher soll bewertet werden, unter welchen Bedingungen bzw. KAs dies möglich ist.

Zuerst sei angemerkt, dass die unberechtigte Ausführung ($\omega = 0.0$) von Quellcode nicht berücksichtigt wird. Dieser Schutz dient primär Systemen mit erhöhten Sicherheitsansprüchen hinsichtlich Security. Laut ISO26262 soll einer Manipulation von Informationen ($\omega = 0.2$) anderer SWCs vorgebeugt werden [89].

Die Kapselung von SWCs bedingt in erster Linie eine Abschottung vor Manipulation des jeweiligen Programmspeichers einer SWC. Daher wird der örtliche Schutz ($\omega = 0.6$) für konsolidierte Systeme priorisiert. Innerhalb der meisten Steuergeräte der Automobilbranche ist keine MMU ($\omega = 0.0$) integriert. Somit wird dieser Zweig nicht weiter betrachtet und der Einsatz einer MPU ($\omega = 1.0$) vorausgesetzt. Der Speicherschutz bzw. dessen Verwaltung stellt für ein sicherheitsrelevantes System eine essentielle Aufgabe dar. Die korrekte Funktionsweise dieser Mechanismen muss zu jeder Zeit gewährleistet werden. Somit werden Mikrokern-basierte ($\omega = 0.4$) und Hypervisor-basierte ($\omega = 0.4$) Systeme einem monolithischen ($\omega = 0.2$) System vorgezogen.

Zeitlicher Schutz ($\omega = 0.2$) bzw. Vorbeugung vor Manipulation während des Informationsaustausches wird gleichermaßen bewertet. Um Fehlern während der Ausführung vorzubeugen, werden Watchdogs ($\omega = 1.0$) eingesetzt. Diese müssen identisch zur Ansteuerung einer MPU von einer zentralen vertrauensvollen Stelle konfiguriert und kontrolliert werden. Ein Watchdog, welcher von der Hardware unterstützt wird, löst im Fehlerfall (je nach Konfiguration) den Neustart eines Systems aus, und beeinträchtigt unter Umständen sicherheitsrelevante Funktionen. Somit werden wiederum

Mikrokern-basierte ($\omega = 0.4$) bzw. Hypervisor-basierte ($\omega = 0.4$) Systeme einem monolithischem System ($\omega = 0.2$) vorgezogen. Auf in Software realisierte Timer ($\omega = 0.0$) und Ablaufkontrollen soll in diesem Szenario nicht zurückgegriffen werden, da diese meist spezifisch von einzelnen Systemkomponenten implementiert werden.

Mechanismen zum Schutz vor Manipulation während des Informationsaustausches ($\omega = 0.2$) werden innerhalb einer Plattform meist von der Hardware bereitgestellt. Daher kann davon ausgegangen werden, dass der Austausch von Informationen zwischen SWCs von Strahlungseinwirkung geschützt ist. Sollen sicherheitsrelevante Informationen über einen Feldbus ausgetauscht werden, wird innerhalb des AUTOSAR Standards auf sogenannte E2E-Bibiliotheken [14] zurückgegriffen, welche eine zusätzliche Prüfsumme ($\omega = 1.0$) zum Informationsaustausch bereitstellen. Dieser Mechanismus kann von allen KAs gleichermaßen ($\omega = 0.33$) umgesetzt werden, da es sich hierbei um eine reine Softwarelösung handelt. Da zu migrierende Systeme in ihrem Funktionsumfang nicht erweitert werden sollen, werden keine zusätzlichen Mechanismen bzw. Redundanzen ($\omega = 0.0$) integriert, um Signale zu plausibilisieren.

	Mikrokern	Monolith	Hypervisor
$\sum \omega$	0.36	0.202	0.438

Tabelle 5.1: Resultate zur Evaluation der Methodik hinsichtlich funktionaler Sicherheit (siehe Abbildung 5.7, 5.6 und 5.8).

Tabelle 5.1 listet die aufsummierten Zweiggewichte der Mikrokern-basierten ($\omega = 0.36$), monolithischen ($\omega = 0.202$) und Hypervisor-basierten ($\omega = 0.438$) Lösungsvariante auf. Hierbei wurden jegliche Gewichte der drei möglichen KAs entlang der Zweige multipliziert und anhand der jeweiligen KAs aufsummiert. Es geht klar hervor, dass durch die projektspezifischen Gewichtungen zur Konsolidierung von Steuergeräten ein Hypervisor-basiertes Architekturkonzept für weitere Analysen verwendet werden soll. Zur Überprüfung der Ergebnisse bildet die Summe der jeweiligen Teilergebnisse $\omega = 1$.

Abgleich der Evaluationsziele

Um die in diesem Kapitel dargestellte Methode zur Bestimmung von optimalen, projektspezifischen Kernelarchitekturen zu untermauern, wurde exemplarisch eine Baumstruktur am Beispiel der NFE funktionale Sicherheit aufgestellt. Eines der Evaluationsziele ist die Anlehnung der vorgestellten Entscheidungsmethode an aktuelle Standards der Automobilindustrie.

Hierfür werden in erster Linie Anforderungen der ISO 26262 [89] (speziell Band 6 - Produktentwicklung auf Softwareebene) aufgegriffen und mit dem aktuellen AUTOSAR Standard abgeglichen. Die genannten Anforderungen zur Gewährleistung der Rückwirkungsfreiheit sind auszugsweise in Abbildung 5.6 genannt und in Abbildung 5.8 bzw. 5.7 verfeinert.

Um die praktische Anwendbarkeit auf spezifische Problemstellungen zu untermauern, wird die Baumstruktur um dargestellte Elemente erweitert. Diese orientieren

sich an klassischen nicht-funktionalen Eigenschaften der Softwareentwicklung [138]. Zusätzliche Elemente werden anhand des in Kapitel 3.4.2 dargestellten Hochintegrationsszenarios ausgewählt und projektspezifisch gewichtet. Wie bereits erwähnt, müssten jegliche Gewichte und Baumstrukturen für anschließende Projektvorhaben angepasst werden. Als resultierende Kernelarchitektur für das in dieser Arbeit darzustellende Hochintegrationsszenario zur Konsolidierung von Fahrzeugfunktionen wurde dabei ein Hypervisor-basiertes Lösungskonzept ausgewählt.

5.7 Zusammenfassung des Kapitels

Innerhalb dieses Kapitels wurde eine Methodik zur Ableitung von Kernelarchitekturen, ausgehend von nicht-funktionalen Eigenschaften, entwickelt. Durch die Erweiterung etablierter Methoden [101, 106] können diese auf die Bedarfe der Automobilindustrie übertragen werden. Somit wurde ein Vorgehensmodell zur Entscheidungsfindung der geeignetsten Kernelarchitekturen (monolithischer, Mikrokern-basierter und Hypervisor-basierter Ansatz) für spezifische Fahrzeugsoftwareprojekte vorgeschlagen.

Die Darstellung erfolgt innerhalb einer hierarchisch unterteilten Baumstruktur (siehe Abbildung 5.2 und 5.4). Dabei werden NFEs in Qualitätsattribute bis hin zu deren Metriken aufgeschlüsselt. Technische Maßnahmen und Fehlerpotentiale wirken sich positiv oder negativ auf die jeweilige Metriken aus. Die Blätter des Baumes bewerten die Umsetzung der jeweiligen technischen Maßnahmen durch eine Kernelarchitektur.

Bäume können an redundanten Teilbäumen zusammengefasst werden. Jegliche Querbeziehungen zwischen komplementär wirkenden NFEs sind bei der Gewichtung zusätzlich zu berücksichtigen. Die Aufstellung eines Entscheidungsbaumes, sowie die Annotation der Zweiggewichte benötigt daher ingenieurmäßiges Vorgehen. Gewichte basieren dabei auf den Erfahrungswerten vergangener E/E-Projekte der Automobilindustrie. Die Gewichtung einzelner Kanten erfordert Expertenwissen und erlaubt eine tendenzielle Beeinflussung der Entscheidungsergebnisse.

Zur Evaluation der Methode wurde exemplarisch ein Entscheidungsbaum der NFE *funktionale Sicherheit* erstellt. Hierfür wurden basierend auf den Lösungsvorschlägen der ISO 26262 [89] relevante Fehlerpotentiale als auch technische Maßnahmen innerhalb einer Baumstruktur aufgeführt. Die gelisteten Baumelemente wurden dabei in Anlehnung an das vorliegende Hochintegrationsszenario gewichtet (siehe Kapitel 3.4.2). Die Auswertung der Zweiggewichte ergab, dass zur Konsolidierung von Fahrzeugfunktionen auf Integrationssteuergeräte der Hypervisor-basierte Isolationsansatz die besten technischen Voraussetzungen für das angedachte Szenario bietet.

Sollen mehrere NFEs in die Auswahl einer geeigneten Kernelarchitektur mit aufgenommen werden, ist die Aufstellung weiterer Entscheidungsbäume notwendig. Dadurch werden Baumstrukturen aller relevanter NFEs eines Projektes mit einbezogen. Weiterhin ist weitläufiges, projektspezifisches Vorwissen notwendig, um die Auswirkungen einzelner Baumgewichte realistischer bewerten zu können.

Wird der erhöhte Aufwand zur Erstellung der Entscheidungsbäume investiert, kann projektspezifisch aus einer Vielzahl von NFEs die geeignetste Kernelarchitektur ermittelt werden. Weiterhin dienen resultierende Baumstrukturen als nachträgliche

Argumentationsgrundlage zur Herleitung der jeweiligen Architektur. Zudem wird die eigentliche Entscheidungsfindung in strukturierter Art und Weise dokumentiert.

Die Güte der Methode richtet sich nach dem investierten Aufwand zur Erstellung des Modells und dem damit zusammenhängenden Detaillierungsgrad der einzelnen Baumstrukturen. Je mehr projektspezifisches Wissen im Modell hinterlegt wird, umso näher führt dies zu einem korrekten Entscheidungsergebnis. Dieses Vorgehen erfordert allerdings erhöhte Arbeitsaufwände zur Erstellung der jeweiligen Baumstrukturen.

6 Kostenanalyse zur Softwareintegration sicherheitsrelevanter Fahrzeugfunktionen

Ziel dieses Kapitels ist die Analyse und Gegenüberstellung der Kosten paravirtualisierter, sicherheitsrelevanter Fahrzeugfunktionen. Hierbei wird sowohl zwischen einer einer monolithischen, Mikrokern-basierten als auch Hypervisor-basierten Systemarchitektur unterschieden (siehe Kapitel 2.1 und 2.2). Es erfolgt die technische Überleitung der verwendeten Architekturprinzipien aus Kapitel 5 zur tatsächlichen Anwendung im Steuergerät. Der Schwerpunkt der nachfolgenden Analyse liegt in der Partitionierung sicherheitsrelevanter AUTOSAR Softwaresysteme, im Zusammenspiel mit den bereits genannten Architekturprinzipien. Dabei sollen die Integrations- bzw. Laufzeitkosten virtualisierter AUTOSAR Steuergeräte näher betrachtet werden. Im Rahmen dieses Kapitels wird, unter Verwendung des exemplarischen Hypervisors der Firma ETAS Ltd., eine Kostenanalyse zur Virtualisierung von Fahrzeugsoftware anhand eines Integrationsbeispiels durchgeführt. Die zur Hochintegration ausgewählten Szenarien sind in Kapitel 3.4.2 und 3.5 erläutert, wobei die Zusammenfassung von Steuergeräten (Szenario C: Fusion) den Kern der Analyse darstellt.

6.1 Herausforderung und Problemstellung

Ein hochintegriertes System konsolidiert eine Vielzahl von Softwarekomponenten und -funktionen mehrerer ECU-Projekte auf einer gemeinsamen Hardware Plattform (siehe Szenarien in Kapitel 3.4.2). Die Konsolidierung von Steuergeräten unterteilt sich in den intrusiven (*engl. intrusive*) bzw. nicht-intrusiven (*engl. non-intrusive*) Ansatz [124]. Beide Methoden sind in Abbildung 6.1 dargestellt und verfolgen unterschiedliche Integrationsansätze.

Im **intrusiven** Ansatz werden SWCs mehrerer Steuergeräte auf einer AUTOSAR BSW integriert. Zur Isolation der Anwendungen werden die Mechanismen des AUTOSAR Standards verwendet. Allerdings birgt die Konsolidierung mehrerer Steuergeräte auf einer gemeinsamen Hardwareplattform unter Verwendung des AUTOSAR Standards diverse technische Problemstellungen. Abhängigkeiten sicherheitsrelevanter SWCs in die BSW sind weiterhin vorhanden. Dadurch können festgelegte ASILs sich auf einzelne BSW Module übertragen (siehe Kapitel 3.3.3).

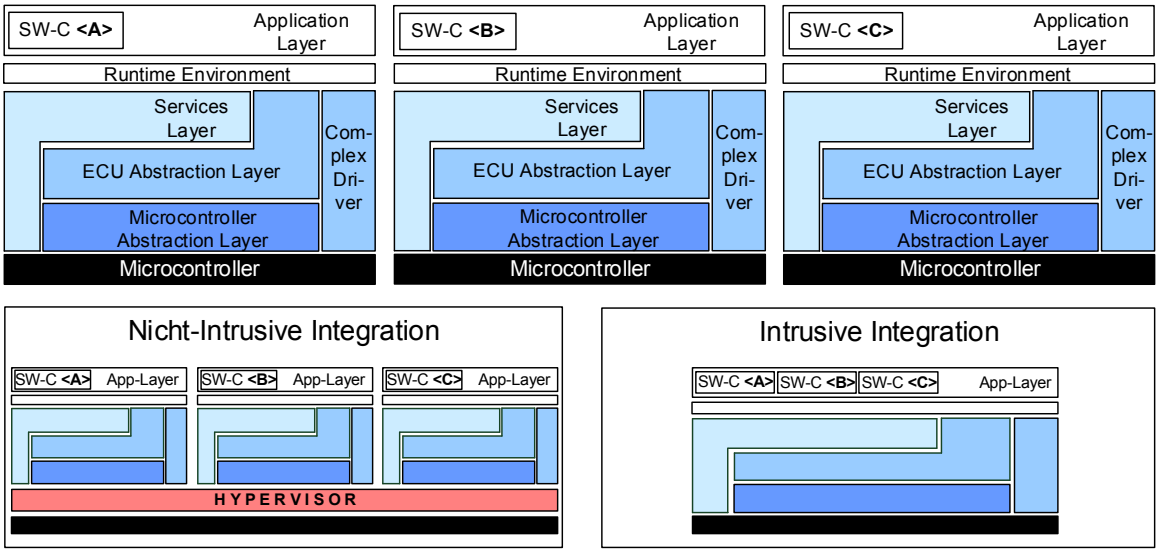


Abbildung 6.1: Intrusive und Non-Intrusive Softwareintegration.

Im **nicht-intrusiven** Ansatz sollen die zu konsolidierenden Steuergeräte und deren BSW so wenig wie möglich verändert werden. Daher werden sie unter Verwendung eines eingebetteten HVs auf einer gemeinsamen Hardwareplattform voneinander isoliert betrieben. Dabei erfolgt keine Verzweigung sicherheitsrelevanter Fahrzeugsoftwaresysteme mit nicht sicherheitsrelevanten Anteilen. Dadurch ist die Vererbung von ASILs zwischen Systemen durch zusätzliche Isolationsmechanismen der VMs eingeschränkt.

Die Migration mehrerer AUTOSAR BSW-Systeme bietet **Implementierungsspielräume**, da Anforderungen fehlinterpretiert werden können. Proprietäre Erweiterungen und BSW-Versionsunterschiede machen eine triviale Verlagerung von SWCs ohne zusätzlichen Portierungsaufwand unmöglich. Je nach BSW Lieferant werden unter Umständen unterschiedliche Ausprägungen bzw. Varianten des AUTOSAR Standards unterstützt.

Weiterhin sind meist nicht alle Fahrzeuganwendungen vollständig auf den AUTOSAR Standard migriert. Somit existiert **proprietäre Software** mit unterschiedlichen Sicherheitsanforderungen, welche in der gemeinsamen BSW als ComplexDriver integriert werden muss.

SWCs mit unterschiedlicher Sicherheitseinstufung können unter Umständen nicht vollständig voneinander entkoppelt werden. Solche Systeme erschweren eine feingranulare Kapselung sicherheitsrelevanter Komponenten und vererben somit ihre Integritätseinstufung auf benachbarte Softwarekomponenten (siehe Abbildung 4.1 und 4.3a).

Ein weiteres Problem stellt die gemeinsame Migration von unterschiedlichen AUTOSAR Systemen bzw. **fremdartigen Betriebssystemen** (wie Linux, Android, etc.) dar. Dies ist bisher mit klassischen Ansätzen des AUTOSAR Standards nicht möglich, da lediglich SWCs auf einer gemeinsamen BSW zusammengefasst werden

können. Eine zusätzliche Bedingung lautet, dass diese Komponenten für die jeweilige BSW Version entwickelt worden sind.

6.2 Verwandte Methoden und Techniken

Die Hochintegration von E/E Fahrzeugsystemen auf einer gemeinsamen ECU ist Forschungsgegenstand (siehe Kapitel 3) und wird bereits von verschiedenen OEMs aktiv vorangetrieben [69, 121]. Der innerhalb der Automobilindustrie gesetzte Standard für E/E Systeme - AUTOSAR in Version 4 [29] - soll nun für den Mehrkernbetrieb verwendet werden [16, 20].

In der ISO 26262 [89] wird gefordert, AUTOSAR SWCs an notwendigen Stellen voneinander zu isolieren. Dies ist notwendig, um sicherheitsrelevante Komponenten innerhalb eines AUTOSAR-Systems betreiben und Rückwirkungsfreiheit garantieren zu können. Um den Qualifizierungsaufwand zu reduzieren, werden hierzu bereits Mikrokern-basierte Ansätze anstelle des aktuellen AUTOSAR OS vorgeschlagen [8, 73, 151].

Eine heterogene Isolation vollständiger Systeme bietet hierbei eine erste Annäherung zur Konsolidierung von sicherheitsrelevanten und nicht-qualifizierten Systemen [48, 121]. Die Notwendigkeit eines Kontrollmechanismus in Form eines HVs wird hierfür in folgenden Publikationen aufgezeigt: [47, 133, 142], als auch vom Autor dieser Arbeit [123]. Darin werden bereits erste Präferenzen für einen paravirtualisierten Lösungsansatz genannt.

HVs wie VMware werden bereits intensiv innerhalb von Rechenzentren eingesetzt [147]. Innerhalb der Avionik- und Automatisierungsdomäne finden bereits verwandte Produkte wie Sysgo's PikeOS Verwendung [93]. Viele Produkte werden für den sicherheitsrelevanten Betrieb genannter Systeme nach relevanten Sicherheitsstandards wie der DO178B [56] bzw. der IEC 61508 [84] qualifiziert. Andere HVs wie Green Hills' Integrity oder Freescales' Topaz sind bekannte Vertreter in diesem Umfeld. OKL4 von OKLabs [76], Xen [34] und XtratuM [104] sind dabei als quelloffene Produkte erhältlich.

Eine Kapselung von sicherheitsrelevanten Funktionen für Echtzeitsysteme wird unter anderem von XtratuM [104] oder Sysgo's PikeOS [93, 112] unterstützt. Jedoch sind diese HVs ausschließlich für Controller verfügbar, welche eine MMU integrieren [47]. Allerdings besitzen viele Steuergeräte innerhalb der Automobilbranchen keine MMU und verfügen lediglich über eine MPU, um hardwareunterstützt Speicherschutz umzusetzen.

Typ-1 HVs wie Xen [33] dienen zur vollständigen Abschottung von Softwaresystemen ohne weitere Host-Systeme. Meistens werden diese nicht für die Zielgruppe kleinerer eingebetteter Systeme entwickelt. Ist ein kleiner Kernel notwendig, wird oft der Begriff des Typ-0 HVs verwendet, welcher die Programmkomplexität auf ein Minimum reduziert und den Großteil des Verwaltungsaufwands innerhalb der VMs verursacht.

Die folgenden Performanz- und Laufzeitanalysen orientieren sich an Veröffentlichungen von Masmano et al. [104] und Carrascosa et al. [52]. Die Messungen für

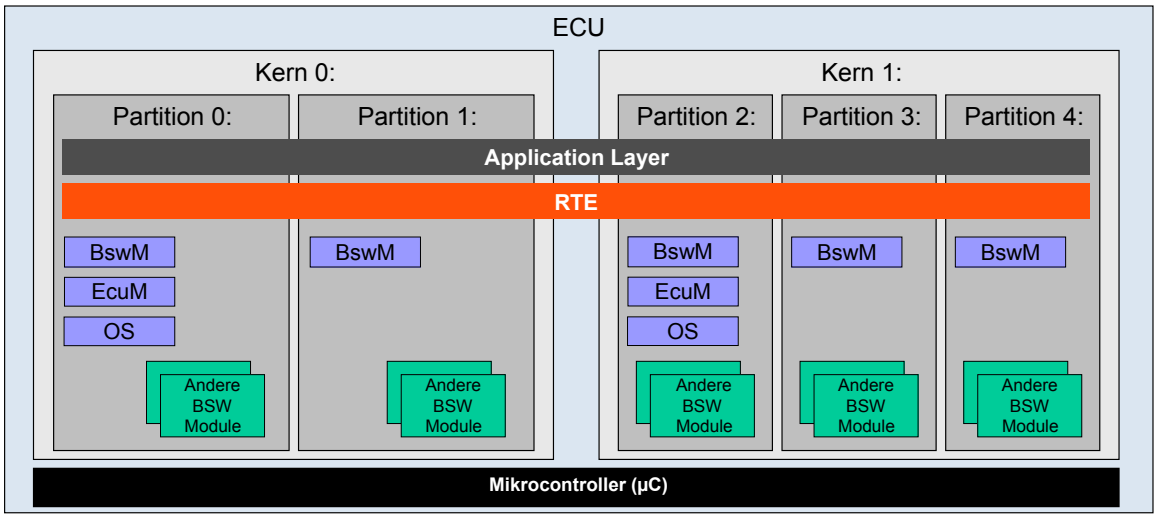


Abbildung 6.2: AUTOSAR Multicore Architektur und deren Softwarepartitionen [123].

Kontextwechsel und Interruptverarbeitung orientieren sich am Vorgehen der Arbeiten von Iqbal et al. [88] und Aichouch et al. [6]. Hamayun et al. [72] liefert hierzu Latenzmessungen für größere Plattformen.

6.3 Monolithische Systemintegration

Ein Steuergerät eines Fahrzeuges, welches über mehrere Rechenkerne verfügt, integriert in seiner Grundkonfiguration ein monolithisches AUTOSAR-System. Dies ist dadurch begründet, dass die AUTOSAR BSW inklusive des OS und allen Treibern innerhalb des MCALs im Kernel-Modus betrieben werden (siehe Kapitel 2.3 und 4.1). Hierbei existiert ein zentrales OS auf dem Hauptkern, welches beim Aufstartvorgang zuerst gestartet wird. Dieses übernimmt alle Verwaltungs- und Steuerungsaufgaben.

Im aktuellen AUTOSAR Standard ist kein symmetrischer Mehrkernbetrieb (*engl. symmetric multiprocessing (SMP)*) vorgesehen, da Kerne und Prozesse (Tasks) statisch zugeteilt werden. Es liegt somit eine asymmetrische Verarbeitung vor (kurz: AMP).

Abbildung 6.2 zeigt die grundlegende Schichtenarchitektur im Mehrkernbetrieb. Pro Partition (bzw. OS-Applikation) wird ein Basic Software Mode Manager (BswM) integriert, um interne Modi zu verwalten. Zur Verwaltung vorliegender Steuergerätezustände wird pro Rechenkern ein ECU State Manager (EcuM) integriert, wobei der EcuM des Hauptkerns alle anderen Satelliten Kerne startet. Im OS enthalten ist der sogenannte Inter-OS-Application Communicator (IOC), welcher sowohl für den Informationsaustausch zwischen Rechenkernen als auch für OS-Applikationen verantwortlich ist [16, 97].

6.3.1 Portabilität und Wartbarkeit

Wie in [11, 26, 28] beschrieben, wird ein Fahrzeug auf logischer Ebene als *System* bezeichnet. Dieses enthält diverse *SystemMappings*, welche den einzelnen Steuergeräten zugeordnet werden. Jedes Steuergerät stellt im AUTOSAR Metamodell eine *ECUInstance* auf technischer Ebene dar. Softwarekomponenten bzw. *SoftwareComponentPrototypes* werden im *SwcToEcuMapping* den einzelnen Steuergeräten zugeteilt. SWCs, deren Schnittstellen bzw. *PortInterfaces* und Kommunikationsports bzw. *PortPrototypes* sind innerhalb des AUTOSAR Standards wohldefiniert [12]. Softwarekomponenten bzw. deren Kommunikationsports werden mittels *AssemblySwConnectors* bzw. *DelegationSwConnectors* verbunden. Durch diese wird die Anwendungsportabilität und Plattformunabhängigkeit von Applikationen zwischen Steuergeräten sichergestellt. Begründet durch mehrere Evolutionsstufen des Softwarestandards entstehen diverse Kompatibilitätsprobleme. Dies betrifft zudem die Wiederverwendung von nicht auf den AUTOSAR Standard portierter Fahrzeugsoftware.

Die Sicherung der ECU-übergreifenden Interoperabilität ist ein Grundpfeiler des AUTOSAR Standards der VFB [13]. Dieses Konzept ermöglicht die Entkoppelung von Softwarekomponenten von deren Hardware und verbindet diese auf einer logischen Ebene. Hierbei wird zwischen dem Signalaustausch mit anderen Steuergeräten und der Kommunikation innerhalb eines Steuergerätes unterschieden. Dabei sollen Softwarekomponenten von der Hardware entkoppelt und aus Entwicklersicht ausschließlich auf logischer Ebene verbunden sein. Innerhalb eines Steuergerätes übernimmt die Funktionalität des VFBs die Schicht der RTE zur Entkoppelung von Applikationssoftware und der Basissoftware (siehe Abbildung 2.3). Softwarekomponenten sind dadurch über Steuergerätegrenzen funktional entkoppelt und können hardwareunabhängig erstellt werden. Diese Unabhängigkeit und die Möglichkeit Software frei zu verlagern ist eine der wichtigsten Anforderungen des AUTOSAR Standards [21].

Typischerweise kann Applikationssoftware während der Integrationsphase in diverser Art und Weise wie in C-Code, C-Objekte oder C-Bibliotheken zur Verfügung gestellt werden. Wie in [28] dargestellt, erfolgt die Integration von Softwarekomponenten über mehrere geordnete Schritte. Hierfür dient das ARXML-Beschreibungsdokument als Eingabequelle für den RTE Generator zur Erzeugung der Applikationsschnittstellen der *RTE Contract Phase*. Auf Basis der daraus resultierenden Schnittstellen muss der Quellcode der Softwarekomponenten erstellt bzw. angepasst werden. Die einzelnen Module der AUTOSAR BSW inklusive des OS können nun konfiguriert werden. Eine ARMXL Beschreibungsdatei dient unter anderen als Informationsquelle zur Konfiguration der RTE, die im Nachgang generiert werden kann.

Um die Integration von Softwarekomponenten in die angegliederte Basissoftware zu erleichtern, können diese unter anderem mit sogenannten *Service Needs* [22] angereichert werden. Diese zeigen alle einzustellende Dienste wie benötigte Speicherblöcke (im NvM Modul), Betriebszustände (im BswM Modul) oder Diagnose Services (im DCM Modul) innerhalb der BSW auf. Wird eine SWC erstmalig dem System hinzugefügt, werden Basissoftwaremodule automatisch konfiguriert. Die genannten *Service Needs* müssen nach standardisierten Richtlinien vom Softwareentwickler spezifiziert sein. Jedoch muss dieses Feature von ausgereiften Integrationswerkzeugen vollstän-

dig unterstützt werden. Ist dies nicht der Fall, müssen fehlende Anteile per Hand konfiguriert werden und bieten zusätzlichen Raum für Fehler.

Eine nachträgliche Verlagerung einzelner Fahrzeugfunktionen und der damit zusammenhängenden SWCs ist aufgrund der logischen Zuordnung in der *System Configuration Description* (System-Beschreibungsdatei in AUTOSAR) möglich. Die Quell- und Zielsysteme müssen innerhalb des *ECU Extracts*, zur Beschreibung der Abhängigkeiten innerhalb des Steuergerätes, iterativ abgestimmt und fehlende Datenverbindungen müssen neu justiert werden. Der detaillierte Prozessablauf der AUTOSAR Methodik und die Erstellung von Softwaresystemen, denen ECUs zugeordnet werden können, wird in [26] und [28] beschrieben. Jedoch erfolgt die finale ECU Konfiguration im seltensten Fall vollständig automatisiert und eine Feinjustierung muss somit manuell vom Integrator vorgenommen werden. Somit können bisher zwar BSW Modulkonfigurationen vollständig exportiert werden, nicht jedoch Konfigurationen, die einzelnen Softwarekomponenten zugeordnet sind. Hierbei muss im Konfigurationsschritt der Basissoftware das betreffende BSW-Modul individuell angepasst werden.

6.3.2 Funktionale Sicherheit

Ein wichtiges technisches Konzept im AUTOSAR Standard zur Isolation auf Anwendungsebene ist die Verwendung von Partitionen. Diese sind aus logischer Betriebssystemsicht den sogenannten OS-Applikationen (siehe Abbildung 6.3) gleichgesetzt [16]. Sie werden auf Taskebene während der ECU-Konfigurationsphase definiert und beinhalten ausgewählte Teile der AUTOSAR-Software. Auf logischer Ebene werden Softwarekomponenten, welche erweiterte Speicherschutzkonzepte bedingen, zusammengefasst und Partitionen zugeordnet. Wie in Abbildung 6.3 illustriert, bedeutet dies aus technischer Sicht, dass einzelne Tasks einer logischen OS-Applikation untergeordnet sind.

Weiterhin werden dort Interrupt Service Routinen (ISRs), Alarme, Scheduling-Tables und Counter zusammengefasst [16, 114]. OS-Applications sind entweder vertrauenswürdig (*engl. trusted*) oder nicht-vertrauenswürdig (*engl. non-trusted*). Vertrauenswürdige OS-Applikationen haben Kernel-Zugriffsrechte auf Betriebssystemfunktionalitäten und somit auf die gesamte eingesetzte Hardware. Partitionen, welche im User-Modus betrieben werden, haben eingeschränkten Zugriff und werden auf Zugriffsverletzungen von einer MPU überwacht. Im Detail sind diese Rechte innerhalb von bestimmten CPU-Registern konfigurierbar. Vertrauenswürdige Partitionen können unter anderem die Register einer MPU verändern oder auch direkt innerhalb fremder Speicherräume Daten modifizieren. Wird ein Fehler festgestellt, können individuelle Rückfallszenarien konfiguriert werden. Hierfür können Partitionen abgeschaltet und gegebenenfalls neu gestartet werden.

Zusätzlich zum Speicherschutzkonzept beschreibt der AUTOSAR Standard den Schutz vor zeitlichen Fehlern. Den jeweiligen OS-Applikationen werden Laufzeitbegrenzungen zugeordnet, um obere und untere Schranken der Ausführungszeiten zu definieren. Durch Partitionierung einzelner Taskgruppen kann deren deterministische Ausführung sichergestellt werden. Ein Timing Fehler tritt auf, wenn innerhalb eines

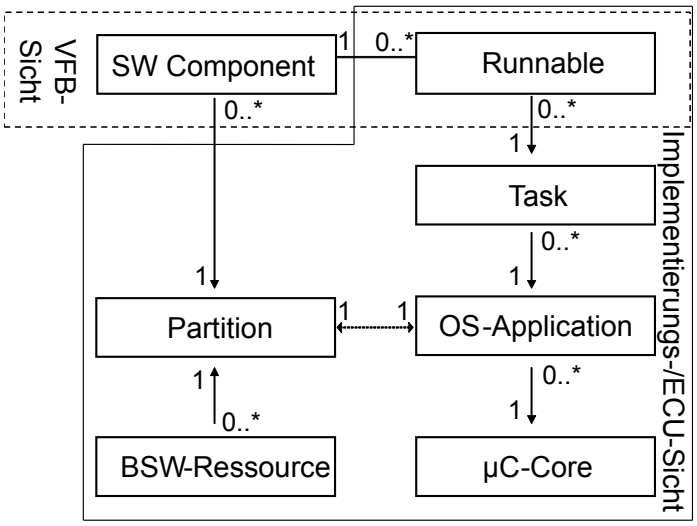


Abbildung 6.3: Logische und technische Sicht zur Kapselung von Systemelementen [123].

Echtzeitsystems ein Task oder Interrupt seine Deadline zur Laufzeit verpasst. Das AUTOSAR Betriebssystem kann auf Fehler im Zeitverhalten einzelner Komponenten reagieren und vordefinierte Maßnahmen einleiten.

Bisher wird vorausgesetzt, dass die BSW oder zumindest das OS Vollzugriff auf alle Hardwareressourcen besitzen. Daher werden lediglich vereinzelte Softwarekomponenten gekapselt und im nicht-vertrauenswürdigen User-Modus betrieben. Der BSW und potentiell sicherheitsrelevanten Applikationen wird der volle Zugriff auf alle Ressourcen gewährt. Diese Vorgehensweise bedingt die Qualifizierung von verbundenen Basissoftwareanteilen nach maximal vorherrschender Sicherheitsstufe [89], welche innerhalb des Steuergerätes bewertet ist (siehe Abbildung 4.1). Um die damit zusammenhängenden Entwicklungskosten in Grenzen zu halten, sollen relevante Anteile der BSW über mehrere OS-Applikationen auf einem Kern voneinander isoliert werden [9, 15, 152].

Neben den genannten Features wie Zeit- und Speicherschutz unterstützt der AUTOSAR Standard diverse Mechanismen, um einen funktional sicheren Systemablauf zu ermöglichen [29]. Der sogenannte End-to-End Kommunikationsschutz garantiert den korrekten Signalaustausch zwischen OS-Applikationen oder gar Steuergerätegrenzen hinweg. Signal-basierte Übertragungsfehler im Bereich der Software und Hardware können mit Hilfe von Error-Detecting-Codes wie Cyclic Redundancy Checks (CRCs) erkannt werden.

6.3.3 Skalierbarkeit

Mehrkernsysteme bieten das Potential, die Skalierbarkeit größerer E/E-Systeme zu steigern und den in Kapitel 3.3.2 genannten Anforderungen gerecht zu werden, um Rückwirkungsfreiheit garantieren zu können. Dabei können Tasks, welche auf verschiedenen Kernen unabhängig verteilt sind, parallel ausgeführt werden (siehe Abbil-

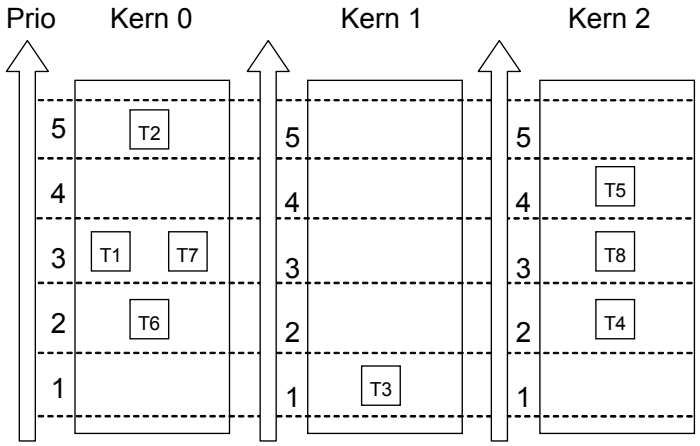


Abbildung 6.4: Task-Allokation innerhalb eines Mehrkernsystems [16, 123].

dung 6.4). Gleichzeitig decken sie den Bedarf an Rechenleistung, welcher womöglich von einem Integrationssteuergerät beansprucht wird. Im Fall von AUTOSAR und dessen OS ist auf jedem Kern ein separates OS integriert, welches sich mit dem Betriebssystem auf dem Hauptkern (*engl. Master-Core*) synchronisiert [16]. Der Hauptkern ist typischerweise der erste Rechenkern, welcher beim Hochstarten des Steuergerätes aktiviert wird. Dieser aktiviert infolgedessen weitere untergeordnete Kerne.

Diese Aufgabe wird vom EcuM des Hauptkerns übernommen, welcher wiederum jeden EcuM der untergeordneten Kerne aktiviert. Zur Vollständigkeit sei erwähnt, dass ein Hauptkern in der Lage ist, auch Kerne zu starten, auf denen ein andere Betriebssystem oder Legacy Code integriert ist. AUTOSAR setzt für den Multicore-Betrieb diverse Restriktionen voraus, welche in [16] gelistet sind (z.B. keine dynamische Zuweisung der Tasks an einzelne Kerne).

Um atomare Softwarekomponenten auf Kerne innerhalb eines Mehrkernsystems verteilen zu können, erlauben *OS-Applications* eine statische Verteilung von Tasks über Kerngrenzen hinweg. Dieser Schritt muss vorab innerhalb der BSW statisch konfiguriert werden. *Partitions* repräsentieren Bereiche, die durch die MPU geschützt werden. Sie sind äquivalent zu OS-Applications eine logische Gruppierung von Softwarekomponenten (siehe Abbildung 6.3). Die Kommunikation zwischen diesen Containern regelt der *Inter-OS-Application Communicator* (IOC), welcher sich innerhalb des Betriebssystems befindet [15, 16]. Daten werden dabei über den gemeinsamen Speicher im System ausgetauscht.

Erfolgt eine Interaktion mit der Hardware wie zu CAN-Controllern oder dem nichtflüchtigen Speicher, so müssen Informationen der verbleibenden CPUs zum Haupt-Rechenkern transportiert und von dort aus kontrolliert abgearbeitet werden. Die BSW wird vollständig auf dem Haupt-Rechenkern betrieben. Um dieses ineffiziente Vorgehen zu optimieren, konzipiert der aktuelle AUTOSAR Standard die Verteilung einzelner BSW-Module [15]. Dies erlaubt einen möglichen Hardwarezugriff auch direkt vom betroffenen Kern ausführen zu können und benötigt keine zusätzliche Synchronisation mit dem Hauptkern. Ein integriertes Modul des BswM wird für jede Partition vorausgesetzt. Dieses verantwortet einen Teil des Fahrzeug- und Applikationszustandsma-

nagements. Das Konzept realisiert eine Client/Server Architektur, deren Umsetzung (Portierung einzelner BSW-Module) jedoch nicht mehr im Rahmen des AUTOSAR Standards verankert ist.

Konsistente Synchronisationsmechanismen ermöglichen der BSW auch den Datenaustausch über mehrere Kerne. Sollen Fahrzeugfunktionen herausgelöst werden, sind jeweilige Systeme neu zu konfigurieren. Ein hochintegriertes System, dass durch mehrere Rechenkerne betrieben wird, ist aufgrund der großen Anzahl an Softwarekomponenten dafür anfällig, häufige Änderungszyklen zu durchlaufen. Der Konfigurationsaufwand und die damit verbundenen Absicherungsiterationen würden zusätzliche Kosten verursachen. Eine Verbesserung auf Seiten der Werkzeuge könnte diesem Problem entgegenwirken. Ein monolithisches System kann daher nur schwer in unterschiedlichen Ausbaustufen betrieben werden. Weiterhin wird eine nachträgliche Integration von neuen Anwendungen erschwert, welche aufgrund rascher Innovationszyklen erst im Nachgang hinzukommen.

6.4 Einsatz eines Mikrokerns

Der Einsatz eines Mikrokerns innerhalb eines AUTOSAR Systems ähnelt stark einer klassischen AUTOSAR Architektur. Daher wird im folgenden Kapitel ausschließlich auf die Unterschiede eines Mikrokern-basierten Systems eingegangen.

Ein Mikrokern besitzt typischerweise eine kleine TCB (siehe Kapitel 2.1). Wird ein Mikrokern eingesetzt, besitzt lediglich ein kleiner Teil (wie Speicherschutz, Taskverwaltung oder Interprozesskommunikation) des Betriebssystems Vollzugriff auf das Gesamtsystem; verbleibende Anteile werden mit User-Rechten betrieben. Abbildung 6.5 zeigt den grundsätzlichen Architekturentwurf einer Mikrokern-basierten AUTOSAR Schichtenarchitektur (siehe Abbildung 2.3 und 4.1). Hierbei werden grüne Anteile im Kernel-Modus und rote Anteile im User-Modus betrieben. Innerhalb sicherheitsrelevanter Funktionen muss somit in erster Linie die kleine TCB des Systems qualifiziert werden. Mikrokern-basierte AUTOSAR Systeme werden bereits von Haworth [73] und Aussagues et al. [8] für sicherheitsrelevante Anwendungen erwähnt und eingesetzt. Durch diese soll lediglich ein kleiner, vertrauenswürdiger Softwareanteil innerhalb des OS (µKernel: siehe Abbildung 6.5, unten links) für weitere Sicherheitsqualifizierungen in Frage kommen [89].

6.4.1 Portabilität und Wartbarkeit

Die Erweiterung eines bestehenden AUTOSAR Systems durch einen Mikrokern verändert nichts an dessen Grundfunktionalitäten zur flexiblen Verlagerung von SWCs zwischen Steuergeräten. Ein Mikrokern wird lediglich zur Erleichterung einer Qualifizierung des AUTOSAR OS und dessen wichtigsten Kernfunktionalitäten eingesetzt.

Allerdings muss hierbei der zusätzliche Integrationsaufwand in Betrachtung gezogen werden, sobald sicherheitsrelevante Applikationen konsolidiert werden sollen (siehe Szenario 3 in Kapitel 3.4.2). Weiterhin sind bereits existierende Sicherungsmechanismen zur Wahrung der Rückwirkungsfreiheit zwischen sicherheitsrelevanten

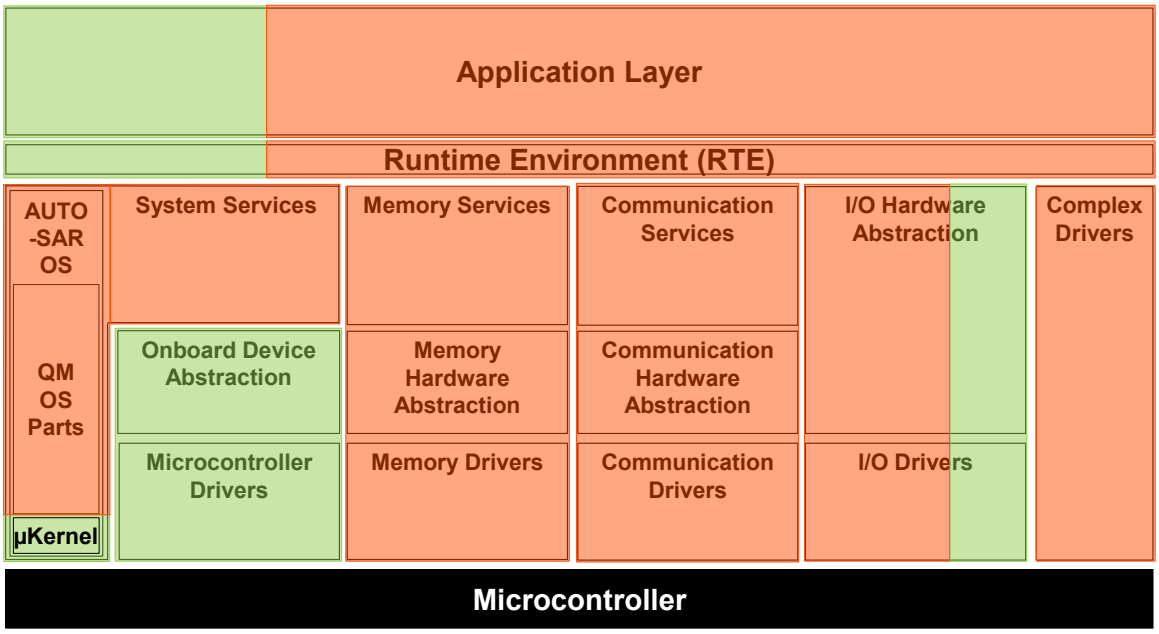


Abbildung 6.5: Grundidee einer Mikrokern-basierten AUTOSAR Schichtenarchitektur [29].

Funktionen nachträglich zu integrieren und abzusichern. Die in Kapitel 4.1 erläuterte Problematik bzgl. der Abhängigkeiten zwischen sicherheitsrelevanten SWCs und nicht-qualifizierten BSW Modulen ist zu beachten.

Das Vorgehen zur Implementierung eines Mikrokerns innerhalb eines AUTOSAR Systems ist in Version 4 des Standards eine proprietäre Erweiterung von BSW Lieferanten [73]. Werden mehrere Systeme zusammengefasst, ist die Vereinheitlichung unterschiedlicher Mechanismen zur Sicherung der Rückwirkungsfreiheit erschwert.

6.4.2 Funktionale Sicherheit

Klassische Softwareanwendungen eingebetteter Fahrzeugsysteme werden weitgehend im Kernel-Modus (Vollzugriff) betrieben (siehe Kapitel 2.3). Aus Sicht der funktionalen Sicherheit ist es zur Reduzierung der Qualifizierungskosten günstiger, die TCB (welche Kernel-Zugriff besitzt) so klein wie möglich zu gestalten. Inklusive des zugerhörigen Kontextwechsels, benötigt der Wechsel zwischen User und Kernel-Modus Rechenzeit und wird daher nach Möglichkeit vermieden. Um diesen Einbußen entgegenzuwirken, wird großen Softwareteilen, wie Hardwaretreibern und vertrauenswürdigen Applikationen, Vollzugriff gestattet. Dies ist unter Verwendung eines Mikrokerns jedoch nicht notwendig. Im Fall eines Mikrokern-basierten AUTOSAR OS soll die TCB auf ein Minimum reduziert werden. Die restlichen Anteile des OS werden getrennt im User-Modus ausgeführt (siehe Abbildung 6.5).

Die Funktionalität des OS beschränkt sich dabei auf die typischen Aufgaben eines Mikrokerns. Diese umfassen sowohl die Interprozesskommunikation, die Speicherverwaltung als auch das Scheduling [145]. Weiterhin wird die MCAL Treiberschicht

weitestgehend im User-Modus betrieben. Bestimmte Hardwarezugriffe benötigen weiterhin Kernel-Rechte, um auf geschützte Register zugreifen zu können (bsp. zur Hardwareinitialisierung).

Funktionalitäten zur Interprozesskommunikation beschränken sich dabei in erster Linie auf die korrekte und verlässliche Funktionsweise der Kontextwechsel im System. Zur Speicherverwaltung soll die Konfiguration und Funktionalität der MPU implementiert werden, um den notwendigen Speicherschutz realisieren zu können. Das Scheduling greift auf die Mechanismen der Interprozesskommunikation (wie Sperr- und Synchronisationsmechanismen) zurück. Dabei soll garantiert werden, dass speziell sicherheitsrelevante Tasks in ihrer Ausführung nicht blockiert werden.

6.4.3 Skalierbarkeit

Die Skalierbarkeit eines AUTOSAR Systems mit einem Mikrokern ist identisch zu einem klassischen Mehrkernsystem. Treiberschichten werden weiterhin ausgehend vom Hauptkern bzw. OS angesprochen. Innerhalb eines Mehrkernsystems erfolgt die Weiterleitung betreffender Anfragen durch die Verwendung des IOC-Mechanismus. Alle Anfragen auf nicht freigegebene Ressourcen bzw. die Ausführung von Kernelgeschützten Instruktionen muss vom OS freigegeben werden. Dies erschwert den Mehrkernbetrieb bzw. erfordert zusätzliche Rechenkapazitäten.

Allerdings ist ein erhöhter Integrationsaufwand für sicherheitsrelevante Fahrzeugfunktionen und deren SWCs nötig. Wie in Kapitel 4 erläutert, sind Abhängigkeiten zwischen der BSW und sicherheitsrelevanten SWCs zu überprüfen. Sollen Dienste der BSW im Kontext einer sicherheitsrelevanten Funktion angesprochen werden, muss deren Funktionalität qualifiziert werden. Um dies zu vermeiden, könnten notwendige Dienste redundant nach den notwendigen Sicherheitsstandards entwickelt und als ComplexDriver integriert werden.

6.5 Systemvirtualisierung

Das Konzept virtueller Maschinen auf Systemebene wurde von Goldberg [62] erstmals erwähnt und definiert [83]. Teure Hardwareressourcen mussten so effizient wie möglich eingesetzt und zur gleichen Zeit den störungsfreien Mehrbenutzerbetrieb garantieren können. Aktuell ist Systemvirtualisierung zur Partitionierung von Systemen ein etabliertes Verfahren in vielen Industriedomänen, wie Mobilfunk oder Avionik [75, 112]. Hierbei existieren unterschiedliche Virtualisierungsansätze und HV Typen (siehe Kapitel 2.2.2). Diese Technologien sollen nun innerhalb der Automobilbranche verwendet werden (siehe Abbildung 6.6).

6.5.1 Portabilität und Wartbarkeit

Die flexible Verlagerung von virtualisierten Systemen auf fremde Hardwareplattformen ist einer der Hauptgesichtspunkte, der den Einsatz eines HVs rechtfertigen. Speziell der Betrieb unterschiedlicher Systeme auf einem gemeinsamen Controller soll durch

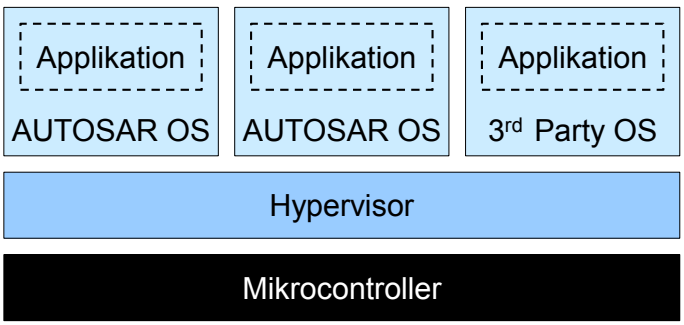


Abbildung 6.6: Partitionierung von Fahrzeugsoftwaresystemen unter Verwendung eines Typ-1 Hypervisors.

virtualisierte Umgebungen ermöglicht werden. Allerdings sind etablierte Virtualisierungskonzepte sowohl für den Serverbetrieb als auch für Desktopsysteme entworfen worden. Diese müssen an die Bedarfe sicherheitsrelevanter Echtzeitsysteme im Automobilsektor angepasst werden.

Ein HV kontrolliert unter anderem privilegierte Hardwarezugriffe. Für den Transfer von Information außerhalb einer VM sind Softwareerweiterungen des HVs verantwortlich. Systemvirtualisierung kann dabei auf unterschiedliche Art und Weise stattfinden (siehe Kapitel 2.2.2). Hierbei wird der Betrieb eines Typ-1 HVs betrachtet.

Generell existieren zwei grundlegende Arten der Systemvirtualisierung [62, 137]:

1. Der Betrieb eines nicht-modifizierten Systems inklusive unverändertem OS (Vollvirtualisierung).

2. Das Grundsystem inklusive dem OS werden an die VMI des HVs angepasst (Paravirtualisierung).

Für Vollvirtualisierung bedeutet dies, dass mehrere unmodifizierte AUTOSAR-Instanzen innerhalb einer ECU unter Verwendung eines HVs betrieben werden. Alle privilegierten und sensitiven Instruktionen werden bei Aufruf innerhalb einer VM abgefangen (durch Auslösung eines Traps) und an den HV geleitet. Paravirtualisierung bedeutet die Modifikation relevanter Registerzugriffe innerhalb einer AUTOSAR-Instanz. Unter Verwendung von Hypercalls werden im Kontext des HVs, privilegierte bzw. sensitive Instruktionen ausgeführt. Weitere Vor- und Nachteile werden in Tabelle 6.1 aufgeführt.

Um Rückwirkungsfreiheit sicherstellen zu können, müssen alle kritischen Operationen innerhalb eines Controllers getrapped und vom HV verarbeitet werden (siehe Kapitel 2.2.1). Wird Vollvirtualisierung verwendet, trappen alle kritischen Instruktionen innerhalb der CPU und werden an den HV weitergeleitet. Ohne zusätzliche Hardwareunterstützung zur Entlastung der Hypervisorschicht, verursacht dieses Vorgehen zusätzliche zeitliche Mehraufwände. Innerhalb von Desktop- bzw. Serversystemen ist diese Hardwareunterstützung bereits integriert (wie Intel VT [86], AMD-V, ARM [108], Freescale [136]). Sind Erweiterungen hinsichtlich Virtualisierung nicht vorhanden, ist Paravirtualisierung eine Möglichkeit, die Intelligenz zur Interpretation

Vollvirtualisierung	Paravirtualisierung
▪ Eine Modifikation des OS und der SWCs ist nicht notwendig. ▪ Die erneute Übersetzung des Systems ist vermeidbar (binary translation). ▪ Die flexible Migration unterschiedlicher Plattformen ist möglich.	▪ Geringe zusätzliche Laufzeitkosten (nahezu nativ) werden verursacht. ▪ Eine schnelle Adaption an andere Plattformen ist möglich. ▪ Die Ermöglichung des Echtzeitbetriebs ist vereinfacht. ▪ Speicher kann ohne MMU abgebildet werden. ▪ Es sind keine speziellen Hardwaretechnologien notwendig.

Tabelle 6.1: Gegenüberstellung von Voll- und Paravirtualisierung für eingebettete Fahrzeugsysteme.

aller kritischen Aufrufe innerhalb des Softwaresystems (wie dem OS) zu vermeiden. Dies ist unter Umständen nicht vollständig möglich, da meist offener Quellcode nicht vorliegt oder aufgrund Lizenzierungsmodellen nicht verändert werden darf.

Weiterhin ist virtueller Speicher eine wichtige Voraussetzung für virtualisierte Systeme. Ist eine MMU im System vorhanden, kann jeder VM ein bestimmter physischer Speicherbereich zugeteilt werden. Hierdurch müssen bestehende Speicherbereiche und Konfigurationen nicht neu übersetzt werden. Das Mapping wird somit hardwaregestützt vorgenommen. Allerdings sind in aktuellen Mikrocontrollern der Automobilbranche weitgehend keine MMUs vorgesehen. Es existieren lediglich MPUs, um Speicherschutzmechanismen zu realisieren. Diese überwacht Speicherzugriffe, nimmt jedoch keine Adressübersetzungen vor.

Das Fehlen einer MMUs innerhalb des Mikrocontrollers wird dadurch begründet, dass sich diese negativ auf den Energieverbrauch auswirkt und Speicherzugriffe verlangsamt. Die monetären Kosten einer MMU sind auch ein weiterer treibender Faktor, um diese nicht auf einem Controller zu integrieren. Allerdings trägt eine MMU essentiell zur flexiblen Verlagerung von virtualisierten Gastsystemen bei. Um trotzdem SWCs ohne tiefere Anpassungen innerhalb von VMs zu kapseln, bestehen zwei Möglichkeiten:

1. Der HV selbst muss die Übersetzung der Adressen vornehmen.

2. Das verwendete OS muss modifiziert werden, um eine nicht-lineare Adressierung zu unterstützen.

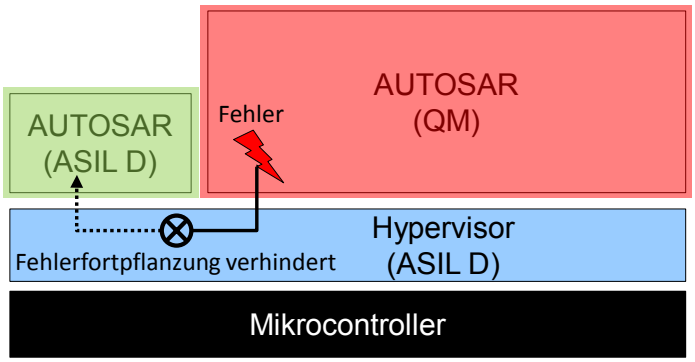


Abbildung 6.7: Integration sicherheitsrelevanter Software unter Verwendung eines Hypervisors.

Alternativ besteht die Möglichkeit, während des Linkens einzelne Kompilate virtualisierter ECUs im Speicher abzulegen.

6.5.2 Funktionale Sicherheit

Um eine großflächige Sicherheitsqualifizierung von Systemanteilen zu vermeiden (siehe Abbildung 4.1), sollen sicherheitsrelevante Applikationen innerhalb von VMs von der nicht-qualifizierten BSW isoliert werden. Da der HV die Softwareschicht mit den höchsten Zugriffsrechten des Systems darstellt (siehe Abbildung 6.7), muss dieser nach dem höchsten integrierten ASIL im System qualifiziert werden. Um diesen Prozess zu vereinfachen, muss die TCB des HVs so klein bzw. transparent wie möglich entwickelt werden. Da jede Komponente innerhalb des HVs Vollzugriff auf das System besitzt, muss jede zusätzlich integrierte Funktion in den HV auf funktionale Sicherheit geprüft werden.

Integrierte Gastsysteme haben im Vergleich dazu eingeschränkten Zugriff auf Hardwareressourcen. Kritische Instruktionen werden bereits von der Hardware abgefangen, wobei diese Eigenschaft eher zur eigentlichen Virtualisierung des Systems verwendet wird.

Es ist davon auszugehen, dass jede Software Fehler enthält [37, 98]. Systematische Fehler können hierbei durch intensive Tests ausfindig gemacht werden und haben ihren Ursprung meist in einer fehlerhaften Spezifikation [89]. Transiente bzw. latente Fehler sind schwer durch Tests ausfindig zu machen und treten meist erst im tatsächlichen Produktbetrieb auf. Wie in Abbildung 6.7 dargestellt, könnten besonders sporadisch auftretende Fehlerbilder innerhalb der zusätzlichen Kontrollschicht des HVs und durch dessen Sicherheitsmechanismen gefiltert werden (siehe Schweizer-Käse-Modell [53]).

Das Ziel eines sicherheitsrelevanten Systems im Automobilbereich ist in erster Linie der Insassenschutz. Der Zusammenhang mit weiteren NFEs wurde bereits in Kapitel 5 erläutert. Somit liefert auch die Zuverlässigkeit bzw. Fehlerfreiheit eines Systems einen notwendigen Beitrag zur Erreichung der Sicherheitsziele in sicherheitsrelevanten Systemen.

6.5.3 Skalierbarkeit

Aufgrund der unterschiedlichen Hypervisortypen, virtualisierte Systeme aufzubauen (siehe Kapitel 2.2.2), wird weiterhin der Ansatz zur Systemvirtualisierung unter Verwendung eines Typ-1 HVs fokussiert. HVs, welche in Rechenclustern eingesetzt werden, sind meist für den symmetrischen Betriebsansatz entwickelt. Hierbei werden einzelne Rechenkerne an unterschiedliche VMs zugeteilt. Weiterhin ist der Betrieb mehrerer VMs auf einem Kern möglich. In beiden Fällen übernimmt der HV die Planung und Zuteilung von Ressourcen.

Um den Echtzeitbetrieb garantieren zu können, existieren bereits diverse Scheduling-Verfahren bzw. Integrationstechniken für VMs [66, 67]. Besonders für den Umgang mit mehreren VMs, verwaltet durch einen Rechenkern, ist eine möglichst nachvollziehbare bzw. berechenbare Planung notwendig. Für eingebettete Systeme ist dieser Betrieb erschwert, da meist zusätzliche Hardwareeinschränkungen berücksichtigt werden müssen. Virtualisierte Systeme besitzen meist kein Wissen über benachbarte Gastsysteme und gehen von exklusiven Rechenressourcen aus. Diese Exklusivität ist nicht vorhanden, sobald sich mehrere VMs die Rechenkapazitäten eines Kern teilen. Besonders für den sicherheitsrelevanten Betrieb müssen virtualisierte Systeme nachvollziehbar ausgelegt sein, um notwendige Qualifizierungsprozesse [89] durchlaufen zu können.

Um den Betrieb mehrerer VMs im sicherheitsrelevanten Umfeld zu ermöglichen, existieren diverse wissenschaftliche Ansätze [66, 67]. Allerdings werden hierfür meist größere Hardwareplattformen verwendet, die aktuell keine Anwendung innerhalb der Automobilindustrie finden. Um trotz dieser Einschränkung VMs auf Kleinststeuergeräten integrieren zu können, ist die Zuweisung einer VM an einen oder mehrere Rechenkerne ein häufig gewählter Ansatz. Damit ist gewährleistet, dass physikalisch Rechenkapazität exklusiv für jedes System vorgehalten ist.

Allerdings skaliert dieses Vorgehen ausschließlich mit der Hinzunahme zusätzlicher Rechenkerne (falls vorhanden). Eine Evolutionsstufe dieser Architektur ist die Integration mehrerer VMs auf einen Rechenkern. Somit können benötigte Rechenbedarfe nicht mehr exklusiv zugeteilt werden und müssen dynamisch vergeben oder innerhalb eines Zeitrasters geplant sein.

Ein weiteres Problem, welches die Flexibilität virtualisierter Systeme einschränkt, sind gemeinsame Zugriffe auf Peripheriecontroller, Speicher, oder andere Hardwareelemente. Darüber hinaus müssen geteilte Hardwareressourcen von einer zentralen Instanz angesteuert und gemeinsame Zugriffe arbitriert werden. Diese Tatsache stellt auch für Serversysteme eine große Herausforderung dar, welche jedoch meist Hardwareunterstützung bzgl. Virtualisierung verfügen. Serversysteme müssen allerdings keine Echtzeiteigenschaften aufweisen und können Lastspitzen mit zeitlichen Verzügen bearbeiten.

6.6 Analyse einer exemplarischen Hypervisorlösung

Wie bereits im vorherigen Kapitel 3 beschrieben wird die Konsolidierung von Steuergeräten unter Verwendung eines HV-basierten Systems angestrebt. Im Rahmen einer Fallstudie wird der Virtualisierungsaufwand und die damit verbundenen Kosten (zeitlich, örtlich und zum Informationsaustausch) eines ausgesuchten HVs analysiert.

Exemplarisch wird hierfür der HV der Firma ETAS Ltd. ausgewählt. Dieser ist auf aktuellen Mikrocontrollern der Automobilsparte lauffähig, welche sich durch besonders kostengünstige Anschaffungspreise aufgrund eingeschränkter Hardwareressourcen auszeichnen. Daher wird eine besonders speicherplatzeffiziente HV-Lösung gewählt. Als Hardwareplattform wird der Infineon TriCore AURIX Mikrocontroller ausgewählt (siehe Kapitel 2.5).

Die HV-Lösung muss weiterhin ohne MMU integrierbar sein. Um Speicherschutz zu realisieren, ist lediglich eine MPU integriert. Softwaresysteme werden meist vor Kompilierung statisch konfiguriert und besitzen somit keine dynamische Speicherverwaltung.

6.6.1 Richtlinien zur Migration virtualisierter Plattformen

Zur Realisierung von virtualisierten Systemen listen Popek and Goldberg [117] bereits grundlegende Anforderungen wie Äquivalenz, Ressourcenkontrolle und Effizienz (siehe Kapitel 2.2.1) auf. Diese Anforderungen sind für eingebettete Echtzeitsysteme in der Automobilbranche mit eingeschränkten Möglichkeiten im Bereich der Hard- und Software nicht immer haltbar. Daher werden im Folgenden Richtlinien und Anforderungen für die Konsolidierung virtualisierter Fahrzeugsteuergeräte aufgelistet.

Wird die nicht-intrusive Wiederverwendung von Software betrachtet, soll der **Portierungsaufwand** einer virtualisierten ECU auf ein Minimum reduziert werden. Der Arbeitsaufwand Steuergeräte zu konsolidieren muss geringer sein, als die Integration ihrer SWCs, auf einer gemeinsamen Hardwareplattform inklusive der Konfiguration der BSW.

Die **Systemperformanz** konsolidierter, virtualisierter Steuergeräte soll nahezu identisch zu nativen Systemen sein. Somit sollen Einbußen durch Hardwareemulation oder die Virtualisierung von anderen Systembausteinen (wie der MPU) vermieden werden. Im Vordergrund steht, dass das spezifizierte Systemverhalten nicht durch Laufzeiteinbußen beeinträchtigt wird.

Das **Kommunikationsverhalten** migrierter Plattformen soll innerhalb der virtualisierten Umgebung nicht verändert werden. Dabei dürfen speziell die Kommunikationsbeziehungen und Austauschformate vorkonfigurierter Feldbusse nicht verändert werden. Die Virtualisierungslösung und deren Schnittstellen muss an die jeweiligen virtualisierten Plattformen angepasst werden. Portierte Software soll daher nahezu unverändert integriert und lediglich durch notwendige Schritte der Paravirtualisierung angepasst werden.

Die **Echtzeitfähigkeit** virtualisierter Systeme soll identisch zu nativen Systemen sein. Daher müssen Eingriffe des HVs vermieden werden. Besonders soll das Kommunikationsverhalten abgebildeter Feldbusse erhalten bleiben. Um Zeitverzüge zu vermeiden, müssen Kommunikationscontroller und deren Treiberschichten direkt miteinander verbunden werden. Eine Aufteilung von Hardwareressourcen ist dabei zu vermeiden.

AUTOSAR ECUs sind typischerweise statisch konfiguriert. Werden diese auf ein virtualisiertes System abgebildet, sollte ein **nicht-intrusiver Integrationsansatz** gewählt werden und die Architektur und einzelne Softwarestrukturen möglichst nicht verändert werden. Allerdings muss die Treiberschicht getauscht werden, falls ein neuer Hardwarecontroller eingesetzt werden soll.

Ein HV darf die **Nachvollziehbarkeit** (Determinismus) konsolidierter Gastsysteme nicht beeinträchtigen. Zur Qualifizierung sicherheitsrelevanter Systeme ist daher eine hohe Transparenz des Gesamtsystems notwendig [89].

Virtualisierte ECUs teilen unter Umständen Speicher, Kommunikationscontroller und eine gemeinsame Crossbar zur Interaktion zwischen Rechenkernen. Innerhalb sicherheitsrelevanter Systeme ist laut ISO 26262 [89] die **Rückwirkungsfreiheit** zwischen Fahrzeugfunktionen und deren SWCs zu gewährleisten (siehe Kapitel 3.3.2 und 3.3.3).

Die Sicherheitsziele von sicherheitsrelevanten Fahrzeugfunktionen dürfen in virtualisierten Umgebungen nicht beeinträchtigt werden. Es müssen die **identischen Sicherheitsziele** erreicht werden, verglichen zum nicht-virtualisierten Fall. Steuergeräte interagieren ausschließlich über gemeinsame Feldbusse (wie CAN oder FlexRay) und weisen keine weiteren physischen Verbindungen auf. Diese Tatsache verringert die Wahrscheinlichkeit, im Fehlerfall eine benachbarte, fehlerfreie ECU zu beeinträchtigen. Virtualisierte Steuergeräte dürfen im Fehlerfall (wie bei einem Systemneustart) andere VMs nicht in ihrer Ausführung beeinträchtigen.

6.6.2 Grundlegender Aufbau des Hypervisors

RTA-HV ist ein von ETAS Ltd. zur Verfügung gestellter Typ-1 HV für paravirtualisierte Systeme. Der HV ist somit direkt auf der Hardware integriert und ist in einer architektonisch tieferen Schicht integriert. Der HV benötigt Kernel-Zugriffsrechte. Die beherbergten Gastsysteme besitzen hingegen eingeschränkte Zugriffsrechte (siehe Kapitel 2.2.2). Abbildung 6.8 zeigt die grundlegenden Funktionen des HVs. RTA-HV ist eine Art Mikrokern (bzw. Nanokern), welcher zudem Hardwareressourcen emulieren kann und selbst kein verwaltendes OS benötigt. Diese Eigenschaften sind mit einem Exokernel vergleichbar [145]. RTA-HV wird direkt auf der Hardware integriert, besitzt Kernel-Rechte und hat vollen Zugriff auf alle Kern- und Peripherieregister. Gast-VMs werden im User-Modus betrieben und haben eingeschränkte Zugriffsrechte. Somit können keine privilegierten bzw. sensitiven Register modifiziert werden.

Alle SWCs verwenden, identisch zu nativen Systemen, standardisierte AUTOSAR Mechanismen. Hierzu zählt sowohl der Zugriff auf BSW-Module als auch die Kommunikation über die RTE-Schicht. Innerhalb eines paravirtualisierten Systems werden

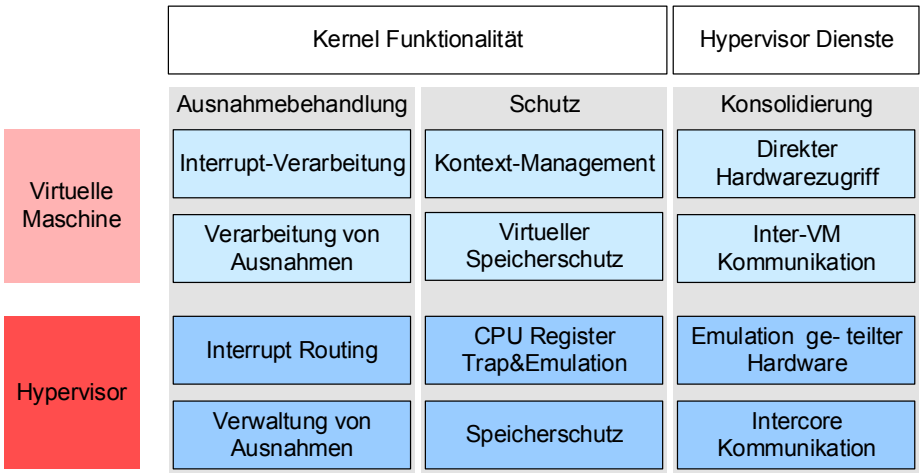


Abbildung 6.8: Grundfunktionen des Hypervisors.

VM-intern Hypercalls verwendet, um Anfragen an den HV zu stellen [137]. Dies betrifft insbesondere den Zugriff auf sensitive bzw. privilegierte Register, die lediglich mit Kernel-Rechten modifiziert werden können. Hierfür sind System-Traps notwendig, um diese Anfragen an den HV zu leiten.

Beispielhafte Hypercalls an den HV sind:

- Eine Modifikation der MPU-Konfiguration, sobald die Anzahl der gegebenen Registerbereiche nicht genügt (siehe Kapitel 2.5).
- Ein Zugriff auf das PSW und das Interrupt Control Register (ICR).

Im aktuellen Aufbau wird jede VM genau einem Rechenkern zugeordnet. Jede VM besitzt dabei exklusiven Zugriff auf den kernlokalen Speicher (siehe Abbildung 6.9). Aufgrund dieser Ressourcenunabhängigkeit ist kein hierarchisches Scheduling zwischen einzelnen VMs notwendig, da Rechenkapazitäten nicht aufgeteilt werden müssen. Interrupts können somit direkt an jeweilige Rechenkerne geroutet werden, wodurch deren Latenz verkürzt wird. Jeder Kern betreibt eine eigene unabhängige HV-Instanz, welche lediglich anhand der Kern-ID unterschieden wird.

Virtual Device Emulators (VDEs) fungieren als frei programmierbare Schnittstellen zwischen den HV-Instanzen, um gemeinsam genutzte Hardware (z.B. Kommunikationscontroller) zu abstrahieren und aufzuteilen. VDEs werden vom HV kontrolliert und in seinem Ausführungskontext im Kernel-Modus betrieben (siehe Abbildung 6.9). Jede VDE Implementierung existiert einmalig innerhalb des Gesamtsystems und jeder Kern bzw. HV-Instanz hat gemeinsamen Zugriff darauf. Der Source Code im HV ist eintrittsinvariant, wodurch kritische Codeabschnitte durch Semaphoren geschützt werden sollten, um Race-Conditions zu vermeiden. Greifen also mehrere Kerne bzw. VMs gleichzeitig zu, muss anhand der Kern-ID zwischen den aufrufenden VMs unterschieden werden. Weiterhin können VDEs Interrupts verarbeiten und andere Rechenkerne unterbrechen. Dies ermöglicht die Implementierung eines Inter-VM-Kommunikationsmechanismus.

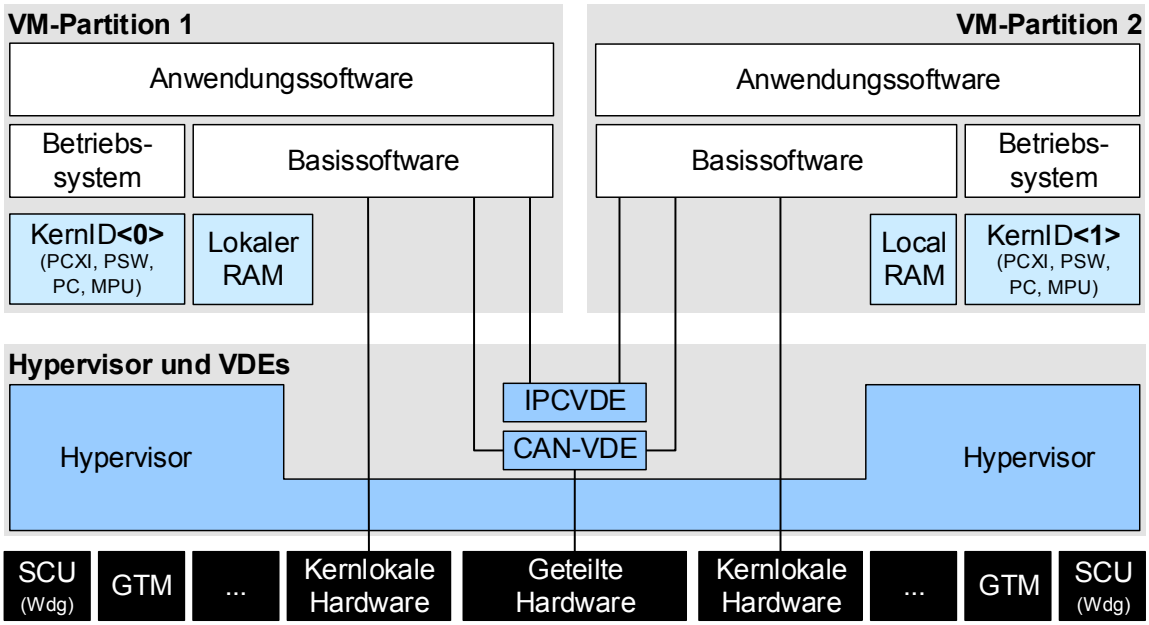


Abbildung 6.9: Schichtenarchitektur eines virtualisierten Systems unter Verwendung der Infineon TriCore AURIX Prozessorarchitektur.

Um einen VDE aufzurufen, ist innerhalb einer VM ein Hypercall notwendig. Dieser ruft unter Verwendung des Schnittstellenaufrufs `HV_CallVirtualDevice(vdeID, *data)` den jeweiligen VDE innerhalb des HVs auf. Als Parameter wird die ID des relevanten VDEs angegeben. Weiterhin wird ein Zeiger auf eine gemeinsame, generische Datenstruktur übermittelt, um Informationen an den VDE übermitteln zu können. Innerhalb des Mikrocontrollers entsteht dadurch ein Class 6 Trap (System Call [2]), welcher als Systemaufruf auf den HV und den jeweiligen VDE verweist. Jeder VDE implementiert eine *request()*-Funktion, in der eingehende Serviceanfragen angenommen werden. Die übertragenen Daten des Hypercalls werden dort abgerufen und verarbeitet. VDEs reservieren die höchsten Interruptprioritäten innerhalb des Systems. Interrupts sind allerdings während der Prozessierung von Traps oder der Ausführung von Quellcode im HV deaktiviert.

6.6.3 Ausnahmebehandlungen von Traps und Interrupts

Eingehende Traps zur Modifikation der MPU-Register müssen vom HV verarbeitet werden. Andere Traps und andere Ausnahmebedingungen können direkt an die jeweilige VM geleitet werden (siehe Abbildung 6.10). Die Verarbeitung von Traps verursacht zeitliche Einbußen zur Laufzeit. Besonders für paravirtualisierte Systeme besteht die Möglichkeit eine Vorkonfiguration der Ausnahmebehandlung und Adresszugriffe vor Kompilierzeit durchzuführen. Diese wird entweder von der Hardware unterstützt (schnell) oder durch den HV ermöglicht (langsam).

Abbildung 6.10 stellt die möglichen Routingpfade für Ausnahmen im paravirtualisierten System dar. Entweder werden Anfragen direkt an eine VM geleitet oder

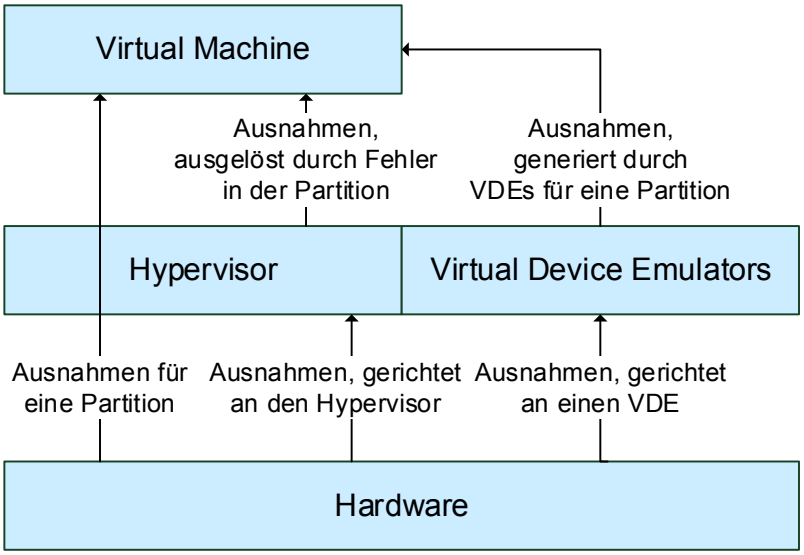


Abbildung 6.10: Ausnahmebehandlungen des paravirtualisierten Systemaufbaus [92].

benötigen weiterführende Aktionen durch den HV. Letzteres betrifft ernsthafte Fehler, die vom HV gehandhabt werden müssen (z.B. Neustarten einer VM oder der gesamten ECU). In jedem Fall werden Interrupts vom HV entgegengenommen und geroutet. Sind Hardwareressourcen direkt an VMs zugewiesen (Direct Device Assignment (DDA)), werden Ausnahmen wie Interrupts direkt an die VM geleitet. Bei geteilten Hardwareressourcen müssen Zugriffe bzw. Ausnahmen einzelner Peripherien von einem zugehörigen VDE gehandhabt werden (z.B. zur weiteren Signalverarbeitung oder -weiterleitung). Interrupts, bestimmt für einen VDE, werden direkt an diesen geleitet. Dieser verwendet dieselbe Interrupt Vector Table (IVT) des HVs und wird in dessen Ausführungskontext betrieben. Weiterhin kann ein VDE einen Interrupt einer VM auslösen, um virtualisierte Peripherien für eine VM zu simulieren.

Der HV selbst reserviert lediglich einen Interrupt mit höchster Priorität (255) pro Kern. Dieser ist mit einem Timer-Modul verbunden und dient als Zähler zur Einhaltung zeitlicher Rückwirkungsfreiheit. Alle verbleibenden Interrupts können frei konfiguriert werden. Tritt ein Interrupt auf, wird dieser vom HV entgegengenommen und anhand der kernlokalen IVT an die paravirtualisierte IVT der VM weitergereicht. Dort findet nach Weiterleitung die typische Prozessierung der ISR statt. Sobald ein Interrupt eine VM erreicht, wird die Interruptpriorität des jeweiligen Prozessors maximiert (höchster verwendeter Interruptlevel innerhalb der VM), um weitere eingehende Interrupts niederer Priorität zu blockieren. Somit wird die Interruptverarbeitung vor Unterbrechungen geschützt (siehe Abbildung 6.13).

Innerhalb eines nativen Systems werden die Interrupts deaktiviert, sobald keine Unterbrechungen zugelassen sind. Im paravirtualisierten Fall ermöglichen Interrupts dem HV zu jederzeit Eingriffsmöglichkeiten in den Ablauf einer VM. Die Laufzeitkosten von Ausnahmeroutinen wie Interrupts werden in Kapitel 6.6.5 näher aufgeführt.

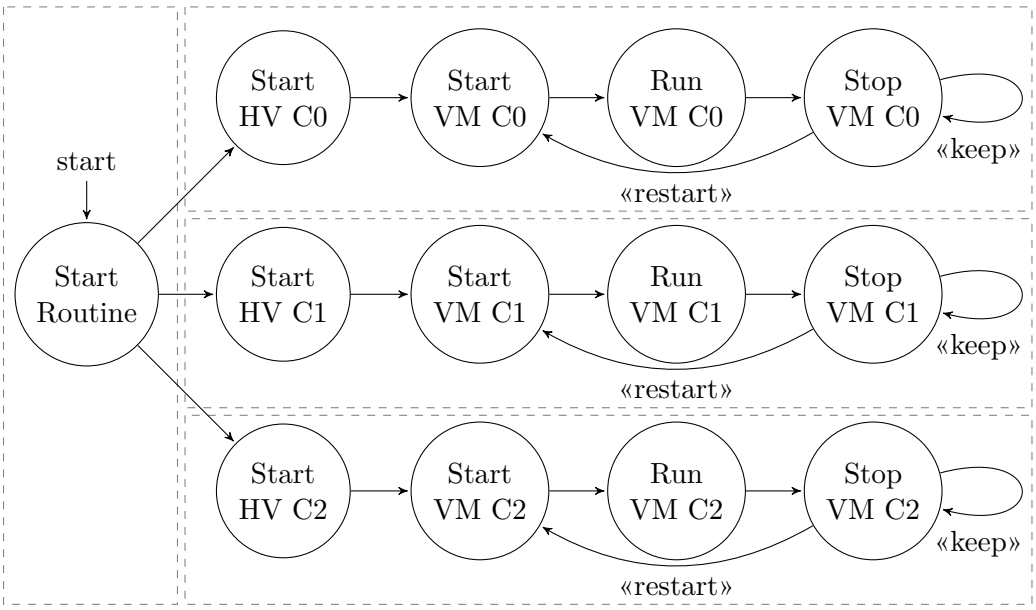


Abbildung 6.11: Systemzustände des Hypervisors und dessen virtueller Maschinen.

6.6.4 Systemzustände und Sicherheitsmodell des Hypervisors

Jede VM wird exklusiv einem Rechenkern zugeordnet. Der Quellcode des HVs ist eintrittsinvariant. Dieser ist daher einmalig im Programmspeicher vorhanden. Es wird alleinig anhand der jeweiligen Identifikationsnummer des Rechenkerns zwischen der jeweiligen HV-Instanz unterschieden.

Abbildung 6.11 zeigt die Systemzustände des HVs und dessen VMs. Das Aufstartverhalten ist ähnlich im Vergleich zum OS im klassischen AUTOSAR Fall [16]. Hierbei starten alle Kerne innerhalb einer gemeinsamen Routine. Dort aktiviert der Hauptkern (Core0) alle weiteren Rechenkerne und initialisiert den jeweiligen Arbeitsspeicher. Im Anschluss wird die eigentliche *main()*-Funktion der jeweiligen HV-Instanzen durch zugehörige Rechenkerne betreten.

Somit wird jeweils eine HV-Instanz pro Rechenerkern als deren zugehörige VDEs gestartet. Zugleich werden die Grundeinstellungen der zugeordneten VM vorgenommen. Dies beinhaltet unter anderem die Konfiguration der MPU zur Gewährleistung des Speicherschutzes und die Anlage der einzelnen Stacks (wie Interrupt-Stack, Context Save Areas (CSAs), User-Stack) [2, 124]. Im nächsten Schritt wird die jeweilige Einsprungadresse der *main*-Funktion innerhalb einer Partition bzw. VM aufgerufen.

Abbildung 6.11 zeigt weiterhin, dass im Fehlerfall einzelne VMs neugestartet werden können. Hierbei wird lediglich die VM selbst neu gestartet (sogenannter Soft Reset). Die Lauffähigkeit der Rechenkerne wird dadurch nicht unterbrochen (siehe Abbildung B.2). Im Fall des AURIX Mikrocontrollers wird beim typischen, hardwareunterstützten Neustart des Systems mindestens jeder Rechenkern inklusive aller Peri-

pheriecontroller neugestartet [5]. Der Neustart des gesamten Systems würde zu einer Beeinträchtigung des rückwirkungsfreien Betriebs zwischen konsolidierten Steuergeräten in VMs führen und muss durch den HV verhindert werden. Der HV entscheidet spezifisch für einen bestimmten Fehlerfall, ob die VM neugestartet werden soll oder nicht. Hierbei wird der Betrieb anderer VMs nicht beeinträchtigt, da lediglich vereinzelte Register des betroffenen Rechenkerns auf ihren Initialzustand zurückgestellt werden. Liegt ein schwerwiegender Fehler vor, kann der Betrieb einer VM vollständig eingestellt werden. Dadurch werden verbleibende Gastsysteme nicht gestört.

Im Folgenden soll dargestellt werden, dass die Entkoppelung von konsolidierten Systemen sich positiv bzw. nicht merklich negativ auf die gesamte Ausfallwahrscheinlichkeit des Systems auswirkt [65]. Typischerweise unterliegt Software keinem Alterungsprozess und besitzt lediglich die statistische Wahrscheinlichkeit (basierend auf Erfahrungswerten), dass ein Fehler bzw. eine Spezifikationsabweichung innerhalb eines bestimmten Zeitraumes auftritt [46]. Dabei errechnet sich der mittlere Ausfallabstand (MTTF) aus der Zuverlässigkeitsfunktion $R(t)$:

$$R(t) = e^{-\lambda t}. \tag{6.1}$$

wobei gilt:

$$\text{MTTF} = \int_0^\infty R(t)\ \mathrm{d}t \tag{6.2}$$

Ist die Ausfallrate im Zeitverlauf jedoch konstant, so ergibt sich:

$$MTTF = \frac{1}{\lambda}. \tag{6.3}$$

Die Ausfallrate λ aus Sicht der Software hängt hierbei von der individuellen Ausfallrate einer virtuellen Maschine $\lambda(\text{VM})$ ab. Diese muss mit der errechneten Ausfallrate des HVs $\lambda(\text{HV})$ addiert werden. Im Fall eines Fehlers wird die Ausführung der restlichen virtualisierten Gastsysteme nicht beeinträchtigt. Somit ergibt sich folgende Gleichung für ein virtuelles Steuergerät (vECU), welches innerhalb einer VM gekapselt ist:

$$\lambda(vECU) = \lambda(\text{VM}) + \lambda(\text{HV}), \tag{6.4}$$

wobei $\lambda(\text{VM})$ die individuelle Ausfallrate einer virtuellen Maschine und $\lambda(\text{HV})$ die statische Ausfallrate des HVs darstellt [65]. Sei N die Anzahl integrierter VMs. Weiterhin errechnet sich die Gesamtausfallwahrscheinlichkeit aller VMs des Systems durch

$$\lambda(System) = \prod_{i=0}^{N}(\lambda(\text{HV}) + \lambda(\text{VM}_i)), \tag{6.5}$$

wobei $\lambda(\text{VM}_i)$ die Ausfallrate der i-ten VM ist.

Im Rahmen der durchgeführten Studie können, unter Verwendung der aktuellen HV-Version werden drei VMs auf dem Infineon AURIX TriCore integriert. Werden mehrere Softwareblöcke bzw. ECUs auf einer Plattform zusammengefasst, können statisch ermittelte Ausfallwahrscheinlichkeiten addiert werden (siehe Gleichung (6.5)).

Durch die Produktbildung der Ausfallwahrscheinlichkeiten zwischen virtualisierten Systemen wird klar, dass sich die gesamte Ausfallwahrscheinlichkeit des Systems verkleinert. Die Fehleranfälligkeit des Gesamtsystems wird ausschließlich durch die fixe Fehleranfälligkeit des HVs verschlechtert. Aufgrund der kleinen TCB eines HVs wird angenommen, dass sich die zu addierende Ausfallrate in Grenzen hält.

6.6.5 Systemperformanz

Im folgenden Kapitel sollen Analyseergebnisse präsentiert werden, um den örtlichen und zeitlichen Mehraufwand durch virtualisierte Umgebungen darzustellen. Die Messungen sind unter Verwendung von RTA-HV in Verbindung mit dem Infineon TriCore AURIX TC27x erstellt worden.

Speicherbedarf

Die Konfiguration des HVs wird vor Kompilierung erstellt und notwendiger Quellcode generiert. Somit ist eine genaue Berechnung des Speicherbedarfs nicht möglich, da Ergebnisse stark von der jeweiligen Konfiguration abhängen. Die dargestellten Messungen repräsentieren eine IST-Aufnahme des verwendeten Testaufbaus.

Das Vorgehen zur Analyse des Hypervisoraufbaus orientiert sich an Masmano et al. [104] und Ripoll et al. [127]. Hierbei beträgt der Verbrauch des nicht-flüchtigen Speichers 45504 Bytes (Text), alle initialisierten Variablen benötigen 48 Bytes (Data) und alle uninitialisierten Variablen verbrauchen 264 Bytes (BSS) Speicher. Ein großer Teil des verbrauchten Speichers wird von den virtualisierten IVTs verwendet, welche pro Kern 2048 Bytes benötigen. Gleichermaßen benötigt die Trap-Tabelle pro Kern 256 Bytes.

Somit besitzt der HV als einzige vertrauenswürdige Systeminstanz einen vergleichbar kleinen Speicherverbrauch. Die TCB ist dadurch klein (ca. 3000 Codezeilen) und bietet das Potential für weitere Optimierungsschritte. Hier muss angemerkt werden, dass dieser Ansatz noch keinen Scheduler für weitere VMs per Rechenkern beinhaltet (siehe Abbildung 6.8).

Laufzeitkosten

Durch Laufzeitmessungen wurde unter Verwendung der AURIX Hardware der benötigte Rechenbedarf innerhalb des virtualisierten Systems ermittelt. Um die Ausführungsdauer bestimmter Instruktionen zu vermessen, wurden Testroutinen erstellt, welche den internen Zähler verwenden. Dessen Frequenz wird mit einem 1:2 Verhältnis zur tatsächlichen CPU-Geschwindigkeit betrieben. Alle Testabläufe wurden sowohl innerhalb eines nativen, als auch eines paravirtualisierten Systems ausgeführt, woraus zwei unabhängige Messergebnisse resultieren.

Alle VMs erhalten direkten, lesenden Zugriff auf relevante Registersätze der Counter, sodass Zugriffszeiten des nativen und paravirtualisierten Systems identisch sind. Aufgrund ausschließlich lesender Zugriffe besteht keine Notwendigkeit zur Überwachung durch den HV. Weiterhin wurden die zeitlichen Mehraufwände für den ei-

gentlichen Messvorgang von den Ergebnissen in Tabelle 6.2 ausgeschlossen. Die in Tabelle 6.2 angegebenen Werte sind Mittelwerte von 100 Messwiederholungen.

Benchmarks	Anzahl der Rechenzyklen		
	Nativ	Zusatzaufwand	Virtualisiert
Zwei nops	2	+50%	3
Setzen des BIV	7	+2914%	211
Vier Schreibvorgänge im PSW	2	+23350%	469
Lesen vom STM_CLC	5	+3180%	164
Setzen eines Bits im Interruptregister	11	+1381%	163
Lesen der CPU-Priorität	5	+160%	13
Setzen der CPU-Priorität	2	+8200%	166
Schreibvorgang zum Interruptregister	6	+2450%	153
Aufruf von ISR	3	+3400%	105
Rückkehr von ISR	13	+107%	27
Aufruf einer Traproutine	26	+284%	100
Rückkehr von einem Trap	11	+18%	13
Setzen des User-Modus in VM	10	+15030%	1503

Tabelle 6.2: Messergebnisse zur Analyse des zeitlichen Mehraufwands innerhalb des paravirtualisierten Systems [122].

In Tabelle 6.2 wird die Anzahl der Rechenzyklen einer Instruktion innerhalb des paravirtualisierten Systems unter Verwendung des HVs dargestellt (siehe Spalte: *Virtualisiert*). Es werden die Zyklen gelistet, welche innerhalb des identischen nativen Systems ohne HV gemessen werden (siehe Spalte: *Nativ*).

Die Zeit zur Modifikation des BIV-Registers ist abhängig von der Zeitspanne, um die IVT Basisadresse einer VM zu setzen. Innerhalb eines nativen Systems wird diese Instruktion sehr schnell vom Prozessor ausgeführt. Im paravirtualisierten Fall ist diese Ausführung wesentlich langsamer. Allerdings werden diese und auch ähnliche Instruktionen lediglich während des Startvorgangs eines Systems verarbeitet. Daher sind die gelisteten zeitlichen Mehraufwände für den eigentlichen Systembetrieb zu vernachlässigen.

Die Manipulation des PSWs und des zugehörigen ICRs benötigt wesentlich mehr Ausführungszeit und wird typischerweise innerhalb des OS modifiziert. Die geringe Performanz resultiert durch diverse Verifikations- bzw. Plausibilisierungsschritte innerhalb des HVs. Aufgrund fehlender Erweiterungen zur Unterstützung von Virtualisierung innerhalb der CPU muss der HV alle Sicherheitsüberprüfungen softwareseitig ausführen. Dies betrifft auch die bereits beschriebene Modifikation der MPU-Register also auch die Konfiguration der Prozessorpriorität (zwecks Unterbrechungsschutz).

Die Modifikation Kernel-geschützter Register muss im Kontext des HVs erfolgen. Hierfür stehen im HV vordefinierte Hypercalls (*HV_ReadUI(), HV_WriteUI(), HV_BitSetUI() und HV_BitClearUI()*) zur Verfügung. Die statischen Einbußen zur Ausführung dieser Hypercalls geben Aufschluss über einen Teil der notwendigen Mehr-

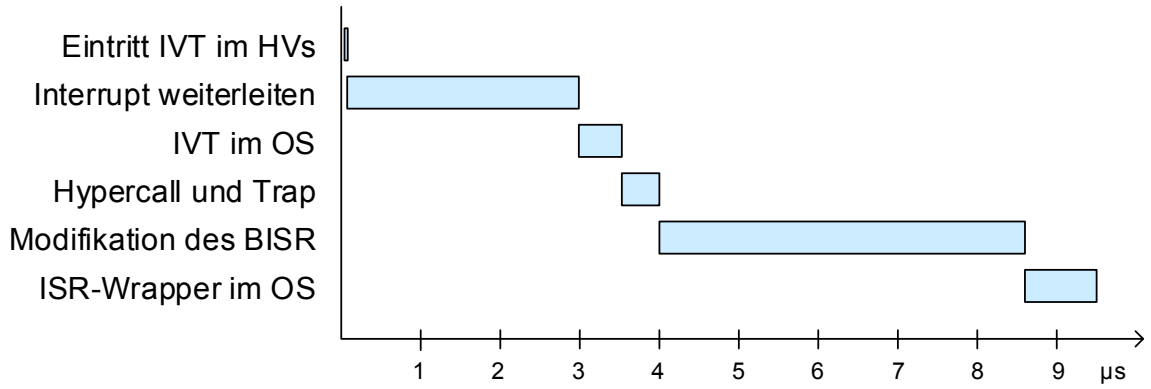


Abbildung 6.12: Prozessablauf zur Weiterleitung von Interrupts der Peripheriecontroller gerichtet an eine virtuelle Maschine [122].

aufwände für I/O unter Verwendung des HVs (z.B. wenn Registerzugriffe ohne Hardwareunterstützung der MPU vom HV geprüft werden müssen). Die zeitlichen Aufwände steigen mit der Menge an potentiell geschützten Adressbereichen an und sind daher konfigurationsabhängig. Die grundsätzliche Ausführungsdauer für Hypercalls ist hierbei statisch, vorausgesetzt es liegen keine weiteren Fehlerfälle vor. Somit können Eventketten einfacher modelliert werden.

Ein ähnliches Performanzproblem tritt auf, sobald die MPU innerhalb einer VM neu konfiguriert wird. Dadurch müssen alle Adressen vom HV auf Richtigkeit geprüft und freigegeben werden.

Laufzeitanalyse der Interruptverarbeitung

Die aktuelle Virtualisierungslösung wurde der Powertrain Domäne angelehnt und auf relevante Anforderungen optimiert. Hierbei sind kurze Interruptlatenzen eine der Schlüsselqualifikationen, um eine größere Anzahl an eingehenden Unterbrechungen (z.B. bei einer Motorsteuerung) in Echtzeit verarbeiten zu können. Die absolute Zeitspanne eingehender Interrupts innerhalb des virtualisierten Systems ist länger als im nativen Fall (siehe Abbildung 6.12 und 6.14). Allerdings ist die relative Interruptlatenz im Vergleich zum zeitlichen Gesamtaufwand eines OS gering bzw. gar vernachlässigbar.

Weiterhin wird angemerkt, dass der Interruptrouter des verwendeten Mikrocontrollers lediglich einmalig im System vorhanden ist. Innerhalb der aktuellen HV-Version ist dieser für integrierte VMs nicht emuliert. Alle Interrupts werden somit vom HV verwaltet. Insbesondere werden eingehende Interrupts an die entsprechenden VMs und deren virtualisierten IVTs weitergeleitet, wodurch zusätzlicher zeitlicher Aufwand entsteht.

Abbildung 6.12 zeigt den Funktionsablauf eines Interrupts bevor dieser innerhalb der jeweiligen ISR verarbeitet wird. Die Messung startet, sobald der Interrupt innerhalb der IVT des HVs angenommen wird und endet sobald das OS den ISR aufgerufen hat. Die gesamte Laufzeit beträgt hierfür statisch 9.5 µs. Diese Laufzeit könnte durch

zusätzliche Unterstützung durch die Hardware (direkte Weiterleitung von Interrupts) deutlich verbessert werden. Zum Vergleich beträgt die Interruptverarbeitung innerhalb eines nativen Systems weniger als 1 µs (siehe Abbildung 6.14 in Relation zur jeweiligen CPU-Frequenz). Die CPU-Frequenz beträgt für diese Messung 80 MHz.

Um den Interrupt innerhalb des HVs weiterzuleiten, werden 3 µs benötigt. Innerhalb des OS einer VM muss die Begin Interrupt Service Routine (BISR) Instruktion ausgeführt werden, um den aktuellen Registerzustand zu sichern und die Prozessorpriorität zu minimieren. Da dieser Vorgang eine privilegierte Instruktion ausführt, trapped diese und wird durch den HV ausgeführt. Die Emulation des BISR ist aufwendig und verursacht somit längere zeitliche Aufwände von zusätzlichen 4.5 µs. Innerhalb eines nativen Systems besteht kein Bedarf, Interrupts von einem HV an einzelne Partitionen weiterzuleiten.

Weiterhin existieren Unterschiede zur Interruptverarbeitung von Category 1 ISR (CAT1) und Category 2 ISR (CAT2) Interrupts innerhalb des OS. Die folgenden Analysen werden unter Verwendung von RTA-OS von ETAS Ltd. durchgeführt. Gemäß dem AUTOSAR Standard benötigen CAT1 Interrupts keine erweiterte Unterstützung des OS [16, 19]. CAT2 Interrupts benötigen zusätzliche Schutzmechanismen des OS, wie zum Beispiel den Unterbrechungsschutz, bevor eine ISR verarbeitet werden kann. Daher ist die Latenz von CAT1 Interrupts typischerweise kürzer. Für paravirtualisierte Systeme sind weiterführende Aufwände für den Schutz von kritischen Anwendungen notwendig.

Unter der Kontrolle des HVs ist die typische Taskausführung und der Zugriff auf Register nicht möglich. Hierfür muss das OS modifiziert werden, um dieses im Benutzermodus ausführen zu können. Die Beeinflussung des HVs oder anderen VMs durch den typischen OS-Betrieb muss in jedem Fall restriktiert werden. Trotzdem müssen virtualisierte Betriebssysteme die Unterbrechung durch HV-Eingriffe zulassen. Somit soll das OS zum Schutz von kritischen Code-Sektionen die Verarbeitung von Interrupts eines Rechenkerns nicht deaktivieren. Ausschließlich die Interrupts, welche an eine VM geleitet werden, sollen blockiert werden. Dies repräsentiert eine elementare Funktionalität innerhalb des paravirtualisierten Systems. Interrupts vom HV sind davon ausgenommen.

Der AURIX ermöglicht die Konfiguration von 255 Interrupts pro Rechenkern [2]. Die ISR ID, welche im Pending Interrupt Priority Number (PIPN) Register des ICRs gespeichert ist, dient gleichzeitig als CPU Priority Number (CCPN). Diese ist individuell für jeden Rechenkern vergeben. Ist die CCPN niedriger als die Interruptpriorität eines empfangenen Interrupts, wird dieser vom Prozessor akzeptiert. Andernfalls wird dieser blockiert. Um nicht-unterbrechbare Coderegionen innerhalb einer VM realisieren zu können, wird die CCPN durch das OS auf die höchst konfigurierte Interruptpriorität einer VM gesetzt. Sowohl alle Interrupts des HVs selbst als auch seiner VDEs besitzen dabei die höchste Interruptpriorität innerhalb des Systems (absteigend von Priorität 255). Die internen Interrupts einer VM werden somit blockiert und Eingriffe durch den HV können in jedem Fall weitergeleitet werden. Allerdings verursacht dieses Verfahren konstante zeitliche Einbußen, um die Prozessorpriorität des CCPN neu zu setzen.

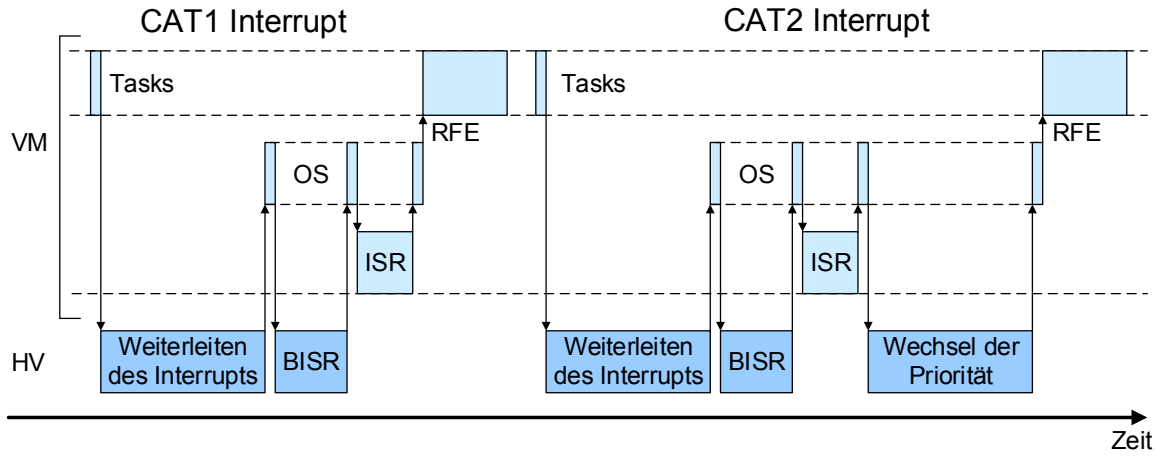


Abbildung 6.13: Ablauf zur Interruptverarbeitung innerhalb virtualisierter Plattformen [64, 92].

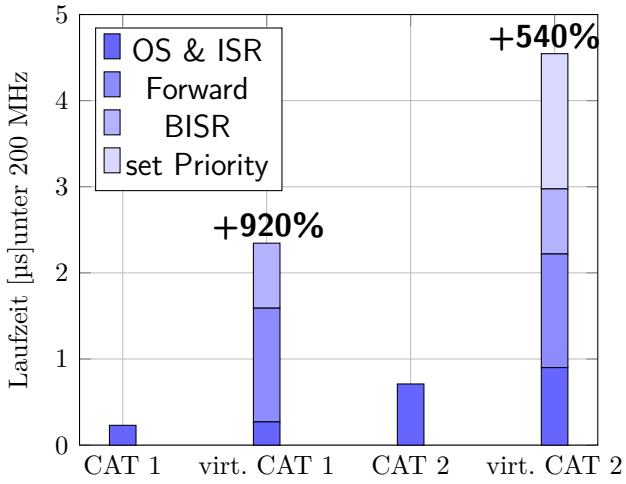


Abbildung 6.14: Interruptverarbeitung und deren Kosten innerhalb virtualisierter Umgebungen [64, 92].

In Abbildung 6.13 wird der grundlegende Ablauf zur Interruptverarbeitung von CAT1 und CAT2 Interrupts dargestellt. Zuerst wird der Interrupt innerhalb des HVs und seiner IVT angenommen. Dort wird dieser vom HV an eine betreffende VM und deren paravirtualisierte IVT weitergeleitet. In einer IVT einer VM wird die jeweilige ISR gespeichert und aufgerufen. Im Gegensatz zu CAT1 Interrupts ist hierbei das OS involviert. Dort entsteht (wie bereits erwähnt) zusätzlicher Kalkulationsaufwand zur Modifikation des BISR. Im Folgenden wird die jeweilige ISR aufgerufen. Während dieser Zeit wird der HV mehrfach durch Hypercalls (durch Class 6 Traps - System Calls) aufgerufen.

Abbildung 6.14 zeigt die Mehraufwände, um Interrupts an die jeweilige ISR zu leiten. Alle dargestellten Messergebnisse sind statisch reproduzierbar und können für eine Worst Case Execution Time (WCET)-Analyse verwendet werden. Wie auch in

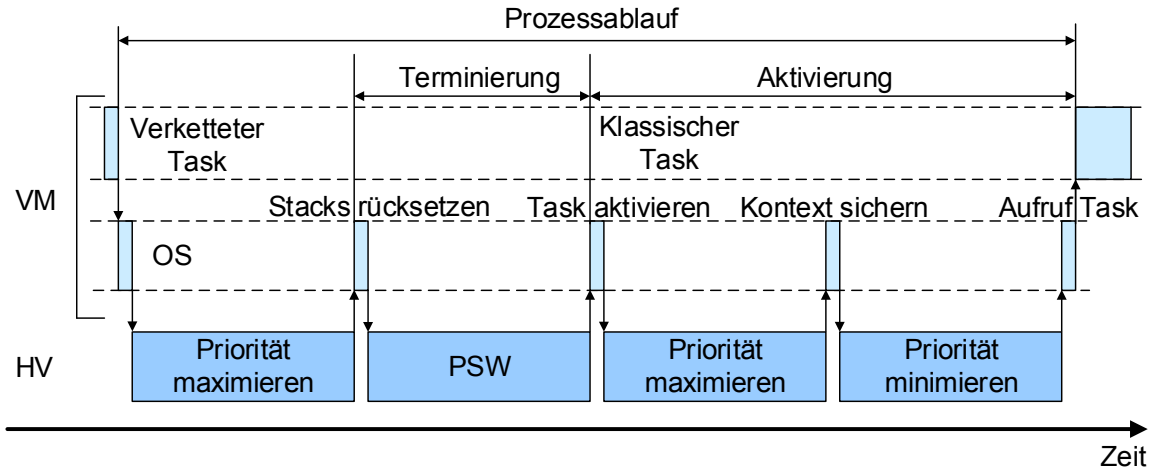


Abbildung 6.15: Ablauf eines Kontextwechsels zur Aktivierung von Tasks innerhalb virtualisierter Umgebungen [64, 92].

nativen Systemen können CAT1 verglichen mit CAT2 Interrupts schneller verarbeitet werden. Die Zeitspanne, einen Interrupt an betreffende VMs weiterzuleiten, beträgt 1.5 µs. Dieser grundlegende zeitliche Aufwand ist identisch für native und virtualisierte Systeme. Auch die Sicherung des unteren Kontextes (unter Verwendung des BISR Befehls) benötigt konstant 1.75 µs. Innerhalb des Verarbeitungsvorgangs einer ISR ist das System unterbrechbar. Um das System zwischen ISR und Return From Exception (RFE) vor Unterbrechungen zu schützen, wird die CCPN maximiert. Diese Modifikation wird vom OS innerhalb der VM vorgenommen und benötigt weitere 0.75 µs.

Analyse der Kontextwechsel von Tasks

Ähnlich zur Ausführung von ISRs, beeinflussen prioritätsbasierte Schutzmechanismen die Ausführungszeiten von Kontextwechseln. Innerhalb eines OS muss die atomare Ausführung während eines Taskwechsels garantiert werden. Zu diesem Zeitpunkt zumindest bis der darauffolgende Task aufgerufen ist, muss die Ausführung vor Unterbrechungen geschützt sein.

Abbildung 6.15 zeigt den typischen Ablauf zur Verarbeitung von Tasks innerhalb des virtualisierten Systems. Im Fall von verketteten Tasks (*engl. chained tasks*) maximiert das OS die CCPN, um resistent bzw. nicht-unterbrechbar gegen weitere eingehende Interrupts zu sein (ausgenommen von Unterbrechungen des HVs). Daraufhin wird der zuvor aufgerufene Task terminiert. Dadurch wird der Stack, also auch die notwendigen CSAs, auf den vorgehenden Zustand zurückgestellt und der nachfolgende, verkettete Task direkt aufgerufen.

Nun wird die identische Aufrufreihenfolge zum Aktivieren eines klassischen Tasks (*engl. activate Task*) verarbeitet. Dort wird wiederum die CCPN maximiert, um keine weiteren Unterbrechungen durch Interrupts zuzulassen. Nachdem der Kontext (z.B. die CSAs) der vorherigen Tasks gesichert ist, wird die CCPN auf ein Minimum re-

duziert und der folgende Task kann verarbeitet werden. Der aktivierte Task ist dabei immer unter der Ausführungskontrolle des OS (aufgrund der jeweiligen Taskpriorität). Er kann nun, aufgrund der minimierten CCPN, während seiner Ausführung von eingehenden Interrupts unterbrochen werden [2].

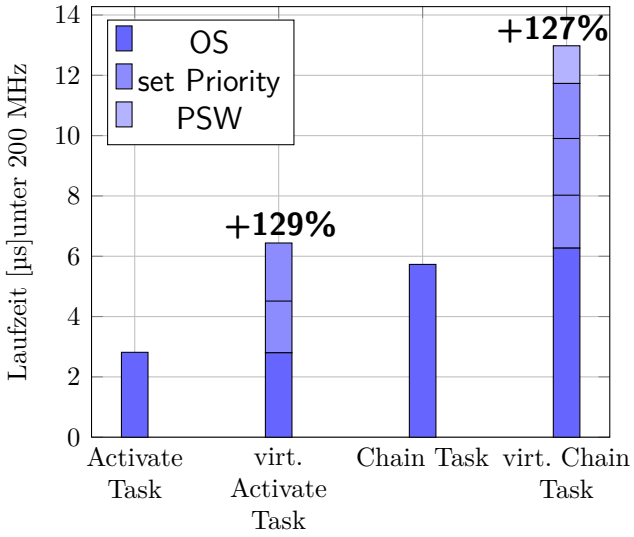


Abbildung 6.16: Aktivierungskosten von Tasks und deren zeitliche Aufwände [64, 92].

Zusammenfassend benötigt jede Veränderung der CCPN zusätzliche 1.5 µs. Die Dauer zur Veränderung von Registern im PSW beträgt 1 µs. Dies ist dadurch begründet, dass Registersätze innerhalb des PSWs lediglich im Kernel-Modus modifiziert werden können. Daher müssen alle Registerzugriffe paravirtualisiert und durch Hypercalls an den HV übergeben werden. Abbildung 6.16 zeigt sowohl die zusätzlichen zeitlichen Bedarfe zur klassischen Aktivierung von Tasks, als auch zur Verwendung von verketteten Tasks innerhalb eines Systems.

Für sicherheitsrelevante Anwendungen unterscheidet der AUTOSAR Standard zwischen *trusted* (im Kernel-Modus) und *untrusted* (im User-Modus) OS-Applikationen. OS-Applikationen integrieren eine Vielzahl von Tasks, die wiederum AUTOSAR Runnables beinhalten [15, 16, 135]. Sie fungieren als Partitionen, um die Grenzen des Speicherschutzes für Anwendungssoftware abzubilden. Daher werden unterschiedliche MPU Adressbereiche des AURIX Mikrocontrollers dazu verwendet, um Vollzugriff für vertrauenswürdige OS-Applikationen und eingeschränkten Zugriff für Tasks in verbleibenden OS-Applikationen zu ermöglichen. Diese Unterscheidung wird typischerweise vom OS kontrolliert.

Innerhalb eines paravirtualisierten Systems verwendet das OS Hypercalls, um im Kontext des HVs zwischen einzelnen MPU Adressbereichen und Registersets zu wechseln. Durch die Verwendung von Hypercalls werden weitere Traps ausgelöst. Um sicherzustellen, dass nicht-vertrauenswürdige OS-Applikationen ausreichend vom restlichen System innerhalb einer VM isoliert sind, verifiziert der HV bei jedem Kontextwechsel eines Tasks zwischen einer vertrauenswürdigen in einer nicht-vertrauenswürdigen Partition bzw. OS-Applikation, ob die Speicherschutzgrenzen kor-

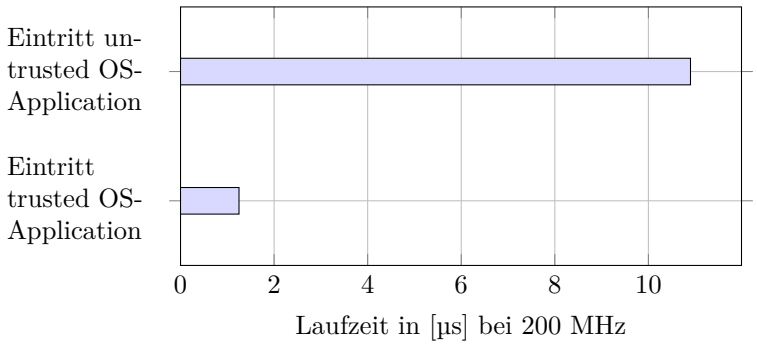


Abbildung 6.17: Zeitlicher Aufwand zum Wechsel zwischen Partitionen im Kernel- und User-Modus [64, 92].

rekt konfiguriert sind. Dies betrifft insbesondere Schreibzugriffe auf die Register von Peripherien oder den Arbeitsspeicher. Diese Verifikation benötigt zusätzliche Rechenzeit und wird in Abbildung 6.17 näher dargestellt.

6.6.6 Portabilität und Wartbarkeit

Um die Echtzeitfähigkeit und Performanz des Systems zu erhalten, werden relevante Teile der integrierten Software paravirtualisiert [47, 133]. Hierfür muss das zu integrierende Steuergerät und dessen Software vorbereitet werden. Da der AURIX TC27x eingeschränkte Unterstützung hinsichtlich Virtualisierung bietet, müssen fehlende Systemeigenschaften durch Softwarelösungen abgebildet werden. Um ausreichende Performanz erzielen zu können, wird das System paravirtualisiert (z.B. Emulation notwendiger privilegierter Instruktionen). Abbildung 6.8 zeigt die grundsätzlichen Mechanismen des HVs, sowie den Umgang mit Traps oder Interrupts. Soll ein Zugriff auf ein privilegiertes Register erfolgen (im Normalfall ausschließlich im Kernel-Modus möglich), wird hierfür die MTCR-Instruktion aufgerufen, die innerhalb des HVs emuliert ist. Somit werden z.B. privilegierte Register im Kontext des HVs ausgeführt, der über die notwendigen Kernel-Rechte verfügt. Ein grundsätzlicher Portierungsaufwand ist an folgenden Stellen notwendig:

- Im Startup Code: Hierbei müssen in erster Linie End-Init (nach Initialisierung gesperrt [4]) geschützte Register paravirtualisiert werden.
- Im Betriebssystem: Innerhalb der Gastsysteme wird ein AUTOSAR OS betrieben. Dieses interagiert mit der Hardware (z.B. mit der MPU). Alle Zugriffe müssen paravirtualisiert werden. Interrupts müssen dabei gesondert behandelt werden.
- Inter-VM-Kommunikation: Ein VDE ist notwendig, um einen Kommunikationskanal zwischen den VMs abzubilden.
- Hardware Treiber: Der AUTOSAR MCAL muss bei Auftreten eines mit Kernel-Modus geschützten Registers paravirtualisiert werden.

Die Initialisierung der Hardware darf ausschließlich von einer Partition bzw. VM vorgenommen werden. Alle folgenden Initialisierungsversuche innerhalb der VMs müssen blockiert werden, um das Überschreiben der initialen Grundkonfiguration zu vermeiden.

Außerdem benötigt das OS zusätzliche Paravirtualisierungsaufwände. Der HV benötigt ständige Kontrolle über alle Gastsysteme. Um mögliche Eingriffe nicht zu verwehren, dürfen die Interrupts der Rechenkerne nicht deaktiviert werden. Um trotz dieser Einschränkung Unterbrechungsschutz gewährleisten zu können, kann die Prozessorpriorität reguliert werden. Sollen bestimmte Programmausführungspfade nicht unterbrochen werden, wird die Prozessorpriorität auf den höchsten im System (VM-intern) befindlichen Interrupt-Level angehoben. Somit werden niederpriore Interrupts nicht angenommen. Nach Beendigung des kritischen Programmbereichs wird die Prozessorpriorität minimiert (Level: 0).

Typischerweise sind Steuergeräte durch ein physisches Busmedium miteinander vernetzt. Diese Verbindungen müssen durch VDEs nachgebildet werden. Diese leiten Nachrichten von einer VM, im Kontext des HVs, an eine andere VM weiter. Ein direkter Signalaustausch zwischen VMs ist im HV nicht implementiert.

Der zeitkritischste Anteil kleiner, eingebetteter Systeme ohne unterstützende Virtualisierungserweiterungen innerhalb des Mikrocontrollers ist die Portierung und Emulation von Treiberschichten und deren Hardwareressourcen [15, 122]. Dies ist der Fall, sobald Ressourcen zwischen VMs geteilt werden. Ist dies nicht der Fall, können Zugriffe direkt an die betreffende Peripherie geleitet werden (DDA). Trotzdem werden alle Zugriffe von der MPU registriert und müssen durch eine Vorkonfiguration genehmigt werden. Genügen vorhandene Adressbereiche der MPU nicht, muss diese zur Laufzeit auf relevante Adressen umkonfiguriert werden oder alle Hardwarezugriffe mittels Hypercalls im Kontext des HVs durchgeführt werden. Dabei muss das native Systemverhalten innerhalb einer VM nachgebildet werden. Der HV verarbeitet alle nicht-freigegebenen Registerzugriffe, welche innerhalb seiner Gastsysteme entstehen.

Typische Szenarien innerhalb eines AUTOSAR Systems sind:

- Lesender und schreibender Zugriff auf Adressbereiche, welche durch die MPU nicht freigegeben sind.
- Ein Wechsel zwischen den Zugriffsrechten innerhalb einer VM (trusted und untrusted OS-Applikationen). Eine Validierung der MPU Einstellungen findet hierbei statt.
- Die Änderung der Prozessorpriorität, um unter Umständen eintretende Interrupts zu blockieren.
- Der direkte Aufruf von VDEs zur Emulation und Arbitrierung von geteilter Hardware.

Abzudeckende Fehlerbilder	Darstellung durch den Hypervisor
Manipulation von Speicher	Der Schutz wird durch Verwendung der kernlokalen MPU sichergestellt.
Lese- oder Schreibzugriff auf Speicher anderer SWCs	Die Zugriffsart (lesend/schreibend) kann individuell innerhalb der MPU konfiguriert werden.

Tabelle 6.3: Unterstützung des Speicherschutzes durch den Hypervisor.

6.6.7 Funktionale Sicherheit

Der HV ist verantwortlich für die Virtualisierung bzw. Emulation der MPU, sodass ein AUTOSAR System innerhalb einer VM ein eigenes Subsystem aus vertrauenswürdigen und nicht-vertrauenswürdigen OS-Applikationen bilden kann. Die Hypervisorschicht ist dabei vollständig von allen VMs entkoppelt. Ein OS innerhalb einer VM kann somit unter Verwendung der Rechenkern lokalen MPU selbst den Speicherschutz zwischen ausgeführten Tasks und deren Anwendungen verwirklichen.

Örtlicher Zugriffsschutz

Die ISO 26262 [89] (Band 6) fordert zur Wahrung der Rückwirkungsfreiheit, dass Speicherbereiche anderer Softwarekomponenten nicht manipuliert werden dürfen. In welcher Art und Weise eine zusätzliche Virtualisierungsschicht dies unterstützen kann, wird durch relevante Anforderungen der ISO 26262 mit dem Leistungsumfang des verwendeten HVs in Tabelle 6.3 abgeglichen. Zur Umsetzung dieser Forderungen appliziert der HV die MPU eines Rechenkerns. Die initiale Konfiguration und Kontrolle der Speicherbereiche erfolgt somit innerhalb der TCB des HVs.

Innerhalb nativer Systeme wird der Zugriffsschutz vom OS kontrolliert und umgesetzt. Dieses unterscheidet zwischen vertrauenswürdigen Partitionen, welche integrierte Applikationen im Kernel-Modus betreiben und nicht vertrauenswürdigen Partitionen, deren Applikationen lediglich eingeschränkte Zugriffsrechte besitzen. Im Falle eines virtualisierten Systems ist der HV die einzige Instanz, welche im Kernel-Modus ausgeführt wird und Vollzugriff im System besitzt.

Jeder Rechenkern im AURIX integriert eine eigene MPU. Diese besitzt jeweils vier Adresskonfigurationen (verknüpft mit 16 Adressbereichen [4]), welche folgendermaßen aufgeteilt sind:

- Konfiguration 0: wird vom HV im Kernel-Mode (SU-Modus im AURIX) verwendet und hat Zugriff auf den gesamten Adressraum.
- Konfiguration 1: eine VM, welche im User-Modus betrieben und deren aktueller Taskkontext als vertrauenswürdig eingestuft ist. Sie besitzt Zugriff auf den gesamten Adressraum der VM.
- Konfiguration 2 und 3: eine VM, welche im User-Modus betrieben und deren aktueller Taskkontext als nicht vertrauenswürdig eingestuft ist. Sie besitzt eingeschränkten Zugriff auf den Adressraum der VM.

Sind die Adressbereiche der MPU nicht ausreichend, kann diese ausschließlich im Adresskonfigurationsset 0 und 1 neu programmiert werden. Diese müssen sich allerdings innerhalb der vordefinierten Adressgrenzen der VM befinden.

Die grundlegenden Aufwände, um Speicherschutz darstellen zu können, entstehen aus einer initialen Konfiguration der MPU während des Systemaufstartvorgangs. Weiterhin entsteht zeitlicher Mehraufwand, um die Konfiguration der MPU innerhalb einer VM während der Laufzeit zu verändern. Die Programmierung der MPU benötigt einen relativ kurzen Zeitraum während des Aufstartvorgangs der ECU [64].

Erhöhter zeitlicher Aufwand ist von Nöten, falls innerhalb der VM die Konfiguration der MPU geändert werden soll. Die Änderung der MPU-Register ist ein schneller Zugriff unter Verwendung eines Hypercalls. Der größte Aufwand wird dabei vom HV selbst verursacht, sobald dieser eine VM im vertrauenswürdigen in den nicht-vertrauenswürdigen Modus wechselt (siehe *Set untrusted mode* in Tabelle 6.2). Es muss ausgeschlossen werden, dass eine nicht-vertrauenswürdige Partition, durch Neukonfiguration der MPU unerlaubten Zugriff auf schützenswerte Ressourcen und Speicherbereiche erhält. Hierfür verifiziert der HV die gesamte Zielkonfiguration von Protection Register Set (PRS) 2 der MPU und vergleicht diese mit erlaubten Adressgrenzen, welche innerhalb der HV-Konfiguration statisch hinterlegt sind.

Zeitliche Schutzmechanismen

Jede VM besitzt exklusiven Zugriff auf einen Rechenkern, wodurch blockierende Zugriffe von weiteren VMs ausgeschlossen werden können. Jedoch besitzt jeder VDE eine höhere Interruptpriorität als integrierte VMs. Um Rückwirkungsfreiheit auf zeitlicher Ebene gewährleisten zu können, muss diese Eigenschaft berücksichtigt werden.

Zur Unterstützung der zeitlichen Schutzmechanismen werden Eigenschutzmaßnahmen implementiert, um den HV in seiner Ausführung nicht zu behindern. Die Anforderungen der ISO 26262 [89] werden mit potentiellen HV Mechanismen in Tabelle 6.4 abgeglichen. Hierbei wird speziell auf Maßnahmen des HVs eingegangen. Alle verbleibenden Maßnahmen werden innerhalb der VM vom AUTOSAR OS verwirklicht.

Um zeitliche Unabhängigkeit garantieren zu können, werden folgende Methoden angewandt:

- Die Prozessorpriorität muss zwischen dem HV bzw. seinen VDEs und einer VM aufgeteilt werden. Dabei ist die Priorität von Interrupts, welche von VDEs behandelt werden, immer höher eingestuft als alle verbleibenden Interrupts, die für eine VM vorgesehen sind. Dadurch wird garantiert, dass durch den HV verursachte Interrupts in jedem Fall berücksichtigt und priorisiert werden.

- Weiterhin werden Interrupts und deren Ankunftszeiten geprüft. Wird eine gewisse Anzahl von Unterbrechungen pro Sekunde überschritten, kann der HV die Annahme verweigern. Andernfalls könnten benachbarte VMs und deren Systeme durch eine zu hohe Interrupt-Last in deren Ausführung gebremst werden.

Jede Peripherieressource des AURIX Mikrocontrollers, welche Interrupts erzeugen kann, besitzt ein Register, mit welchem der jeweilige Zielkern und die Priorität

Abzudeckende Fehlerbilder (nach ISO26262)	Darstellung durch den Hypervisor
Ausführungsblockierung	VMs auf Rechenkernen greifen individuell auf VDEs zu. Wird eine Semaphore bzw. Mutex nicht freigegeben, kann dies durch externen Watchdog sensiert werden.
Deadlocks	Die Verwendung von Watchdogs und Timer, um nicht freigegebene Semaphoren zu sensieren. Innerhalb der VM reguliert das AUTOSAR OS die Ausführung.
Livelocks	Ausführungskontrolle erfolgt innerhalb von VDEs: die Ausführung von Interrupts zwischen VMs ist innerhalb des HVs reguliert (z.B. DOS-Attacke). Bei Überschreitung wird eine Ausnahmeroutine ausgeführt.
Falsche Allokation von Ausführungszeit	Dies wird innerhalb der VM durch das AUTOSAR System umgesetzt.
Falsche Synchronisation zwischen Softwareeinheiten	Dies wird innerhalb der VM durch das AUTOSAR System umgesetzt.

Tabelle 6.4: Unterstützung des zeitlichen Schutzes durch den Hypervisor.

des Interrupts festgelegt werden kann. In einem nativen System wird dieses Register innerhalb des AUTOSAR OS bzw. im Rahmen des MCALs kontrolliert. In einem virtualisierten System stellen diese Register ein Problem dar. Sie sind weiterhin unter der Kontrolle des virtualisierten OS und des MCALs. Tritt dort ein Konfigurationsfehler auf, werden Interrupts fehlgeleitet und beeinträchtigen einen Rechenkern. Somit kann eine Isolation auf zeitlicher Ebene nicht garantiert werden. In der verwendeten Version von RTA-HV steht der Interrupt-Controller selbst nicht unter der Kontrolle des HVs.

Schutz während des Informationsaustausches

Der Schutz während des Informationsaustauschs wird in der ISO 26262 [89] besonders berücksichtigt. Der HV verhindert jede direkte Interaktion zwischen VMs, ausgenommen der Befähigung, Interrupts an andere Kerne zu richten. Diese Eigenschaft muss in folgenden Ausbaustufen des HVs berücksichtigt werden.

Die ISO 26262 [89] fordert zusätzliche Schutzmaßnahmen zur Vermeidung von Fehlerbildern, die während des Informationsaustausches auftreten können. Die geforderten Punkte werden dem Verhalten und den Mechanismen des HVs in Tabelle 6.5 gegenübergestellt.

Prinzipiell werden vom AUTOSAR Standard bereits diverse Mechanismen zur Absicherung gegen Informationsmanipulation während des Signalaustausches zwischen SWCs und ECUs zu Verfügung gestellt (siehe AUTOSAR E2E-Library [14]). Für den Informationsaustausch zwischen VMs in E/E-Systemen der Fahrzeugindustrie ist kein allgemeingültiger Standard vorhanden.

Abzudeckende Fehlerbilder (nach ISO26262)	Darstellung durch den Hypervisor
Verlust bzw. Verzögerung	Der Informationsaustausch zwischen VMs ist im Vergleich zu nativen Systemen (ein gemeinsames AUTOSAR System) verzögert. Der Verlust eines Signals ist nur dann möglich, sobald die ausgelösten Interrupts nicht mehr verarbeitet werden können.
Wiederholung, Hinzunahme von Datenpaketen	Dies wird HV-intern zwischen einzelnen VMs kontrolliert, muss jedoch innerhalb jeder VM zusätzlich abgesichert werden.
Maskierung bzw. falsche Adressierung von Botschaften	Dies wird HV-intern zwischen einzelnen VMs kontrolliert, muss jedoch innerhalb jeder VM zusätzlich abgesichert werden.
Korruption von Nachrichteninhalten	Zwischen VMs unterstützt die Hardware durch Prüfsummen, um Übertragungsfehlern entgegenzuwirken.
Blockierende Zugriffe	Jede VM kann gleichermaßen auf einen VDE des HVs zugreifen. Dabei gehaltene Semaphoren müssen jedoch nach einer gewissen Zeitspanne wieder freigegeben werden (blockieren auch andere Rechenkerne).
Asymmetrische Übertragung, bzw. unterschiedliche Empfänger	die Übertragung von Signalen zwischen VMs wird durch den HV abgesichert. Asymmetrische Übertragung kann in VDEs implementiert werden. Die Ansteuerung erfolgt jedoch ausgehend von einzelnen VMs.

Tabelle 6.5: Unterstützung des Informationsschutzes durch den Hypervisor.

6.6.8 Skalierbarkeit

Im Folgenden werden die Analysen zur Bildung virtualisierter E/E-Umgebungen zusammengefasst. In erster Linie sollen innerhalb der Betrachtung AUTOSAR-basierte Systeme stehen, welche auf ein gemeinsames Steuergerät konsolidiert werden sollen. Es soll beurteilt werden können, ob die dargestellte Lösung zur Hochintegration einer Vielzahl von virtualisierter Steuergeräten geeignet ist. Speziell der MCAL der AUTOSAR BSW steht hierbei als hardwarenahe Treiberschicht im Vordergrund (siehe Kapitel 2.3). Dabei werden grundsätzliche Portierungsschritte aufgeführt.

Integrationsszenario hochintegrierter virtualisierter Systeme

Innerhalb der dargestellten HV-Lösung werden VMs exklusiv einem Rechenkern zugewiesen (siehe Kapitel 6.6.2). Somit ist die Anzahl der integrierbaren virtualisierten Steuergeräte auf die maximale Anzahl zur Verfügung stehender Rechenkerne beschränkt. Dies setzt voraus, dass alle Rechenkerne eine identische Prozessorarchitektur inklusive deren ISA besitzen.

Es sollen somit nach Möglichkeit keine hardware-technischen Abhängigkeiten zwischen den Rechenkernen vorherrschen. Dies garantiert einen zeitliche und örtliche, unabhängigen Betrieb aller integrierten Systeme. Grundlegende Hardwarebausteine wie Timer, MPUs oder kernlokaler Arbeitsspeicher müssen vorhanden sein. Dies wird vom verwendeten Mikrocontroller (Infineon TriCore AURIX) innerhalb dieser Arbeit zur Verfügung gestellt.

Jede VM integriert weiterhin ein eigenständiges OS. Wie in Kapitel 6.6.6 erläutert, muss der Betrieb in einer paravirtualisierten Umgebung sichergestellt sein. Schutzmechanismen und Kontextwechsel werden weiterhin vom OS verwaltet, jedoch im Kontext des HVs angesprochen. Innerhalb dieser Arbeit sind diese Bausteine unabhängig für jeden Rechenkern vorhanden und verursachen somit keine Emulationsaufwände.

Allerdings greifen alle Rechenkerne und die darauf befindlichen VMs auf einen gemeinsamen Programmspeicher zu. Dieser besitzt innerhalb der aktuellen Variante des AURIX Mikrocontrollers zwei Speicherbänke, welche bei jedem Schreibvorgang kurzfristig für andere zugreifende Teilnehmer gesperrt sind. Dies kann zu Verzögerungen führen und müsste bei zukünftigen Controller-Ausbaustufen berücksichtigt werden [54, 64]. Weiterhin ist, wie bereits in Kapitel 6.6.5 angemerkt, der Interrupt Router lediglich einmalig im System vorhanden und alle Interrupts müssen somit vom HV verwaltet und weitergeleitet werden.

Sollen virtualisierte Steuergeräte nach wie vor mit externen Busteilnehmern interagieren, muss auf bestehende Kommunikationscontroller oder anderweitige Peripheriegeräte zurückgegriffen werden. Diese sind lediglich in begrenzter Anzahl vorhanden und müssen im Falle einer Mehrfachverwendung für jede betreffende VM emuliert werden.

Allgemein müssen während der Portierung eines bestehenden Systems in eine paravirtualisierte Umgebung die Zugriffe auf geschützte Register durch Hypercalls gekapselt werden. Diese dürfen ausschließlich vom HV erfolgen und treten verstärkt innerhalb der Treiberschicht des AUTOSAR MCALs auf. Somit entsteht Paravirtualisierungsaufwand, welcher für jedes einzelne virtualisierte Steuergerät zu investieren ist.

Integrationsanalyse am Beispiel einer minimalistischen AUTOSAR Treiberschicht

Zur weiteren Analyse des Paravirtualisierungsaufwandes sollen ausgewählte Treibermodule des AUTOSAR MCALs betrachtet und für eine spätere Portierung in eine VM analysiert werden [15]. Hierfür werden die MCAL Treiber GPT, DIO, CAN, MCU, WDG und FLS näher betrachtet (siehe einzelne MCAL-Treiber in Abbildung 4.1).

Innerhalb des AURIX Mikrocontrollers besitzt jeder der 3 Rechenkerne exklusiv einen Watchdog-Timer, welcher durch das **WDG**-Modul konfiguriert wird. Wird der Ablauf eines Watchdog-Timers festgestellt, muss der Neustart des gesamten Systems verhindert werden, da dieser alle VMs betrifft (siehe Abbildung B.2). Alleinig im HV sind die jeweiligen Richtlinien zum tatsächlichen Neustart des Systems und dessen VMs hinterlegt. Grundsätzlich sollten daher alle Watchdogs unter Kontrolle des HVs stehen.

Werden Kommunikationscontroller angesteuert, sollten diese nach Möglichkeit exklusiv zugeteilt werden. Ist dies nicht der Fall, muss ein VDE speziell für diese Aufgabe implementiert werden. Dieser muss die jeweilige Vorkonfiguration der Kommunikationsinterfaces vornehmen. Im Fall von **CAN** müssen die individuelle Bitrate und unterschiedlichen Nachrichtenobjekte berücksichtigt werden. Dabei müssen alle Register zur Initialisierung des CAN-Controllers paravirtualisiert werden. Diese sind während des Betriebs nicht beschreibbar und lediglich während der Initialisierungsphase mittels Kernel-Rechten konfigurierbar. Für sicherheitsrelevante Systeme muss einer fehlerhaften Konfiguration vorgebeugt werden. Speziell für Bussysteme (wie CAN oder Flexray) muss die Interaktion zwischen virtualisierten Steuergeräten aufrechterhalten bleiben, da diese auch Botschaften zwischen einzelnen Partitionen austauschen könnten. Die Übermittlung von Nachrichten erfolgt zusätzlich über den virtualisierten CAN-Treiber. Dieser arbitriert eingehende und ausgehende Botschaften an den physischen Feldbus und tauscht CAN-Frames auch zwischen VMs aus.

Eingehende bzw. ausgehende Signale via Ports und deren PINs werden durch *General Purpose I/O* Schnittstellen des AURIX Mikrocontrollers gelesen. Diese werden durch das **DIO**-Modul für Digital Input/Output angesteuert. Wiederum ist ein exklusiver Hardwarezugriff anzustreben, um Emulationsaufwände zu vermeiden. Grundsätzlich können Ports durch unterschiedliche Adressierung zwischen VMs aufgeteilt werden. Es entsteht Emulationsaufwand, sobald einzelne PINs zwischen VMs aufgeteilt werden müssen. Die Adressauflösung der vorliegenden MPU ist hierfür zu grob und Adressbereiche einzelner PINs sind zu dicht aufeinander folgend [4]. Eine feingranulare Abschottung durch die MPU ist somit nicht möglich. Daher muss auf die Verwendung eines VDEs zurückgegriffen werden, in welchem jeweilige PIN-Belegungen hinterlegt sind und den jeweiligen VM-Zugriffen zugeordnet werden. Sind Steuergeräte konsolidiert und PINs exklusiv zugewiesen, muss unter Umständen deren finale Belegung neu zugeteilt werden.

Ähnlich zu den Kommunikationsschnittstellen werden *General Purpose Timer* durch das **GPT**-Modul angesteuert. Es ist dabei zu beachten, dass bei hohen Timer-Frequenzen, welche durch den HV an dessen Partitionen weitergereicht werden, ein großer Rechenaufwand innerhalb der HV-Schicht selbst entsteht. Daher sollte dieser Treiber nach Möglichkeit exklusiv bzw. statisch an Rechenkerne zugeteilt werden. Sind also nicht ausreichend Timer vorhanden, muss diese Zuteilung dynamisch durch den HV erfolgen. Diese Zuteilung ist allerdings mit starken Querbeziehungen zu I/Os verbunden, weshalb eine klare Aufteilung nur schwer lösbar ist. Weiterhin benötigen Partitionen konfigurationsabhängig unterschiedliche Timer Einstellungen (Divisoren und Frequenzen).

Der nicht-volatile Datenspeicher (*engl. flash*) muss speziell für Schreibzugriffe sequentiell angesteuert und geschützt werden. Hierfür ist das **FLS**-Modul verantwortlich. Lesende Zugriffe sind immer erlaubt. Allerdings sind diese nur dann möglich, soblad der Datenspeicher nicht durch einen gleichzeitigen Schreibvorgang blockiert ist. Sollen mehrere Informationen abgelegt werden, muss dieser Vorgang sequentiell mit zeitlichen Abständen erfolgen. Wird der Flashspeicher beschrieben, ist dieser während des Schreibvorgangs gesperrt, sowohl für Lese- als auch weitere Schreibzugriffe.

Um zeitgleiche Zugriffe zu vermeiden, müssen Hardwaretreiber für den Datenspeicher paravirtualisiert und unter Umständen innerhalb eines VDEs integriert werden.

Innerhalb der AUTOSAR BSW ist das **Microcontroller Unit (MCU)**-Modul für die hardwarespezifische Initialisierung des Controllers verantwortlich. Innerhalb jeder Partition bzw. VM wird eine Hardwareinitialisierung vorgenommen, da dieses Treibermodul in jeder BSW vorhanden und konfiguriert ist. Dieses ist unter anderem für die Konfiguration der Taktfrequenz der Prozessoren (*engl. clock trees*) verantwortlich. Problematisch ist die Tatsache, dass Partitionen, welche nicht-intrusiv konsolidiert werden, dabei unterschiedliche CPU-Taktraten anfordern. Zur Auflösung dieser Problematik muss die initiale Hardwarekonfiguration von einer zentralen Stelle übernommen werden. Entweder geschieht dies innerhalb einer VM oder durch einen VDE. Erfolgt die Hardwareinitialisierung innerhalb einer VM, ist anzunehmen, dass die Konfiguration des MCU-Treibers abzuändern ist. Dies widerspricht allerdings dem nicht-intrusiven Integrationsvorhaben, die BSW Konfiguration so wenig wie möglich zu verändern. Erfolgt die Hardwareinitialisierung innerhalb eines VDEs, muss die Ansteuerung der MCU-Treiber innerhalb der VMs deaktiviert werden.

Gegenüberstellung der Integrationskosten

Die in Kapitel 6.1 genannten Anwendungsfälle zur Integration von Fahrzeugsteuergeräten auf einem gemeinsamen Mehrkernsystem soll nun gegenübergestellt werden. Dabei wird angemerkt, dass Konfigurationsaufwände für HV und BSW im Rahmen dieser Arbeit nicht messbar sind und je nach Projektgröße und -komplexität variieren.

Die vorgestellte Lösung zeigt, dass eine Konsolidierung virtualisierter Steuergeräte unter Verwendung eines HVs möglich ist. Die unter anderem in Tabelle 6.2 aufgeführten Laufzeitkosten sind begründbar, und können durch zusätzliche Hardwareunterstützung optimiert werden. Der Portierungsaufwand wurde in Kapitel 6.6.8 zusammengefasst. Dieser beschränkt sich, für den Fall, dass keine Hardware emuliert werden muss, auf den Paravirtualisierungsaufwand innerhalb des AUTOSAR MCALs.

Soll Software konsolidiert werden, existieren 2 Möglichkeiten welche mit unterschiedliche Integrationskosten verbunden sind: Die **Migrationskosten** zur Konfiguration der AUTOSAR BSW und RTE, um eine Vielzahl von Softwarekomponenten zu vereinen (Methode: *intrusive*) und die **Portierungskosten** auf einen HV (Methode: *non-intrusive*) [122].

Werden mehrere AUTOSAR Systeme vereint, entsteht Konfigurationsaufwand der RTE, der BSW und des MCALs (siehe Kapitel 6.3). Die Wiederverwendung von Funktionalitäten innerhalb der AUTOSAR Schichtenarchitektur wirkt sich somit positiv auf den Speicherverbrauch aus. Werden AUTOSAR und nicht AUTOSAR-basierte Systeme zusammengeführt, entsteht typischer Portierungsaufwand auf den jeweiligen AUTOSAR Standard. Dieser variiert je nach Systemgröße und ist nicht Teil dieser Arbeit. Weiterhin ist eine erneute Konfiguration der RTE, der BSW und des MCALs notwendig.

Wird Legacy Software auf einen HV portiert, entsteht in erster Linie Paravirtualisierungsaufwand für nicht erlaubte Registerzugriffe innerhalb des AUTOSAR MCALs (siehe Kapitel 6.6.8). Weiterhin wird ein paravirtualisiertes OS benötigt. Muss Peri-

pherie zwischen VMs geteilt werden, sind VDEs notwendig, um relevante Zugriffe zu kontrollieren. Hierbei entsteht allerdings kein Wiederverwendungseffekt von statischen Speicher (RAM und ROM), da dieser in jeder Partition neu angelegt werden muss.

Um eine Paravirtualisierung des verwendeten AUTOSAR OS vorzunehmen, sind sowohl unerlaubte Registerzugriffe, als auch die Handhabung von Schutzmechanismen für kritische Codeabschnitte zu berücksichtigen (siehe Kapitel 6.6.6). Registerzugriffe innerhalb des MCALs können innerhalb der MPU freigegeben werden. Speziell innerhalb des eingesetzten AURIX Controllers müssen End-Init geschützte Register paravirtualisiert werden. Diese werden allerdings in erster Linie zur Modulinitialisierung aufgerufen und verursachen somit zur Laufzeit keine weiteren Effizienzeinbußen.

6.7 Systembewertung

Im Rahmen dieser Arbeit sollen Steuergeräte zusammengefasst werden. Wird dabei angenommen, dass darauf entwickelte Fahrzeugsoftware an den AUTOSAR Standard angelehnt ist, können SWCs einfacher migriert werden. Die zuvor aufgeführten Architekturprinzipien (Monolith, Mikrokern und Hypervisor) sollen nun konzeptionell beurteilt und bewertet werden. Im Fokus steht Szenario 3 aus Kapitel 3.4.2 zur Fusion von mehreren ECUs. Abbildung 6.18 beleuchtet die nicht-funktionalen Eigenschaften Wartbarkeit, Skalierbarkeit, Effizienz, Portabilität und funktionale Sicherheit näher.

6.7.1 Effizienz

Effizienz versinnbildlicht in dieser Arbeit die Performanz eines Systems und den damit verbundenen Ressourcenverbrauch integrierter Software, im Vergleich zu dessen Geschwindigkeit zur Laufzeit.

Im Rahmen dieser Arbeit dienen AUTOSAR-basierte Systeme als Benchmark hinsichtlich Geschwindigkeit und Ressourcenverbrauch. Besonders durch Wiederverwendungseffekte im Rahmen der Schichtenarchitektur und durch eine toolgestützte Codegenerierung können diese Systeme performant ausgelegt werden. Allerdings bestehen starke Abhängigkeiten zu benötigten Konfigurationsswerkzeugen der BSW. Generierter Quellcode erreicht nicht die Performanz bzw. Ressourceneffizienz von manuell erzeugtem Code. Daher erhalten AUTOSAR-basierte Systeme keine maximale Ergebnisbewertung.

Nahezu identische Ergebnisse können für Mikrokern-basierte AUTOSAR Systeme erzielt werden. Allerdings ist dort eine erhöhte Anzahl von Kontextwechseln notwendig, um zwischen User und Kernel-Modus umzuschalten. Die meisten Hardwaretreiber innerhalb der MCAL-Schicht werden im User-Modus betrieben. Die Vielzahl an Kontextwechsel beansprucht stärker einzelne Rechenkerne, woraus ein schwächeres Ergebnis im Vergleich zum klassischen AUTOSAR Ansatz resultiert. Hinsichtlich des Ressourcenverbrauchs ist anzunehmen, dass dieser identisch zu einem klassischen AUTOSAR System ist.

Paravirtualiserte Systeme profitieren durch eine Vorabmodifikation des Quellcodes, speziell im OS bzw. in der Treiberschicht. Dadurch wird die Komplexität des

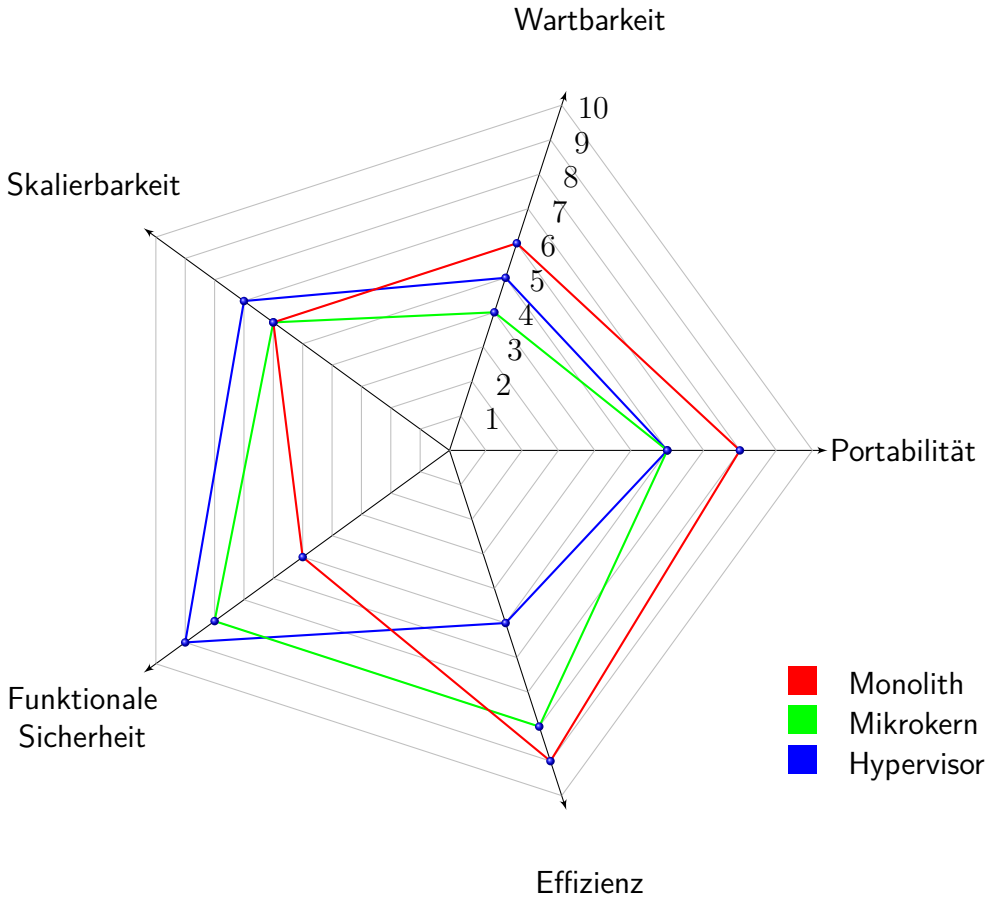


Abbildung 6.18: Gegenüberstellung zur Bewertung der Architekturprinzipien: Monolith, Mikrokern und Hypervisor.

HVs verringert. Aufgrund von Hypercalls werden sensitive Instruktionen nicht weiter im Kontext einer VM ausgeführt, wodurch sich die Performanz hinsichtlich des Laufzeitverhaltens verbessert. Werden mehrere AUTOSAR-basierte Systeme zusammengeführt, müssen allerdings grundlegende Elemente (wie statischer Code, globale Variablen, etc.) innerhalb der BSW mehrfach in einzelnen VMs integriert werden. Allgemeine Softwareteile können nicht innerhalb eines Gesamtsystems wiederverwendet werden. VMs stellen geschlossene Systeme dar, wodurch für jede VM erneuter Integrationsaufwand von Basisfunktionen notwendig ist. Dies stellt einen Mehraufwand in Bezug auf den Verbrauch von Speicher dar. Allerdings gilt dies nicht für Speicherbedarfe von funktionsabhängigen Elementen (lokale Variablen, statischer Quellcode der Applikationen, etc.). Dieser Verbrauch fällt auch bei einer Zusammenführung von mehreren AUTOSAR Systemen auf eine gemeinsame BSW an. Weiterhin ist innerhalb der Fallstudie in Kapitel 6.6.5 dargelegt, dass sich die Systemperformanz aufgrund virtualisierter Interrupts bzw. Kontextwechsel verschlechtert. Im Vergleich zum HV-Konzept ist das klassische AUTOSAR System effizienter, hinsichtlich des Rechnenzeitbedarfs.

6.7.2 Portabilität

Die Portabilität eines Systems soll innerhalb der aktuellen Betrachtung ausdrücken, wie leicht ein System auf eine gemeinsame Plattform überführt werden kann.

Eine der Kernfunktionalitäten von AUTOSAR ist die Erleichterung zur nachträglichen Austauschbarkeit von Hardware und der BSW. Somit bietet ein AUTOSAR System aufgrund seiner Beschaffenheit gute Voraussetzungen um nachträgliche Veränderungen flexibel aufnehmen zu können. Allerdings bedingt ein Wechsel des Grundsystems erneute Konfigurationsaufwände der BSW, da BSW-Lieferanten proprietäre Konfigurationswerkzeuge verwenden und ein konventioneller Austausch an Moduleinstellungen meist nicht möglich ist. Weiterhin werden meist nicht alle geforderten Funktionsumfänge des AUTOSAR-Standards umgesetzt und lediglich projektspezifisch implementiert.

Werden Mikrokern-basierte Systeme betrachtet, stellen der Austausch von Funktionen und die Migration auf andere Plattformen zusätzliche Hürden dar. Mikrokernbasierte Lösungsansätze sind im aktuellen AUTOSAR Standard (Version 4.2.1) nicht spezifiziert und werden von einzelnen Herstellern zusätzlich als Lösungskonzept angeboten. Ein einfacher Wechsel ist aufgrund der proprietären Lösungsvariante nicht reibungslos möglich, da somit der AUTOSAR Standard nicht mehr vollständig erfüllt wird. Allerdings entstehen hierbei keine weiteren Probleme, sobald die Treiberschicht aufgrund eines Controllerwechsels getauscht werden muss. Die Kernfunktionalität des Mikrokerns beschränkt sich hierbei auf das OS und der Kapselung von Ressourcen.

Der Aufwand um existierende Systeme in virtualisierte Umgebungen zu portieren ist gering (siehe Kapitel 6.6.6). Kapitel 6.6.8 zeigt hierbei speziell die Schritte zur Portierung der Treiberschicht auf. Diese werden vom Hardwarelieferanten zur Verfügung gestellt und sind aufgrund von Qualifizierungsmaßnahmen meist offen einsehbar. Allerdings ist die Paravirtualisierung eines AUTOSAR OS für den Endnutzer nicht ohne weiteres durchführbar. Meist ist der Quellcode, selbst nach Abschluss der Systemkonfiguration, nicht offen einsehbar und geschützt. Weiterhin müssten Paravirtualisierungsschritte nach jedem Update des OS oder einzelner Hardwaretreiber wiederholt werden. Daher sollte die Anpassung dieser Schichten bestenfalls vom Lieferanten selbst erfolgen. Jedoch muss dieser die jeweilige API bzw. VMI des gewünschten HVs unterstützen, um eine Interaktion zwischen OS und dem HV zu ermöglichen. Es wird angenommen, dass die Mikrokern-basierte Lösung eine identische Bewertung eines HV-basierten Integrationskonzeptes aufweist.

6.7.3 Wartbarkeit

Unter Wartbarkeit wird die nachträgliche Produktpflege und Unterstützung von Updates eines Systems verstanden.

Zusammengeführte AUTOSAR Systeme stellen hierfür ein in sich geschlossenes Mehrkernsystem dar. Sollen Funktionen nachgeladen bzw. zusätzliche Funktionalitäten von unterschiedlichen Zulieferern integriert werden, ist der Abstimmungsaufwand erhöht, da meist große Teile des Softwaresystems angepasst werden müssen. Allerdings stellt hierfür AUTOSAR ein standardisiertes Verfahren zur Verfügung [29].

Für Mikrokern-basierte Systeme hingegen ist anzunehmen, dass diese sicherheitsrelevante Funktionen bereits integrieren (siehe Kapitel 6.4). Um Rückwirkungsfreiheit garantieren zu können, müssen Schutzkonzepte erweitert werden, wodurch sich der Test- und Absicherungsaufwand erhöht. Dies ist aufgrund von unterschiedlichen Querbeziehungen zwischen SWCs und der BSW nicht immer möglich. Weiterhin ist der Mikrokern-basierte Lösungsansatz noch nicht im AUTOSAR Standard verankert, wodurch kein klarer Integrationsprozess vorhanden ist. Im Vergleich zu einem virtualisierten System besteht auch keine Möglichkeit, neue Funktionalitäten unabhängig betreiben zu können. Dies erschwert den Integrationsprozess auf eine neue Technologieplattform.

Die HV-Lösung wird hierbei stärker gewichtet, da einzelne VMs kleinere, abgeschlossene Systeme darstellen. Dies verringert einerseits die Komplexität des Gesamtverbundes und andererseits den Test- und Absicherungsaufwand auf Softwareebene, da nicht betroffene VMs unverändert integriert werden können. Allerdings bestehen starke Abhängigkeiten zum HV und dessen Funktionsumfang, wodurch der HV Ansatz einem klassischen AUTOSAR System unterliegt.

6.7.4 Funktionale Sicherheit

Funktionale Sicherheit wurde bereits tiefer in Kapitel 3 betrachtet. Im Rahmen dieser Bewertung werden Schutzmechanismen zur Gewährleistung der Rückwirkungsfreiheit zwischen SWCs fokussiert. Dabei soll speziell die Fehlerausbreitung innerhalb des Systems durch dedizierte Kapselungsmechanismen betrachtet werden.

Der AUTOSAR Standard hingegen verfügt über diverse Mechanismen zum Schutz sowohl auf zeitlicher und örtlicher Ebene, als auch zur Gewährleistung der Manipulationssicherheit während des Informationsaustauschs. Im betrachteten Hochintegrationsszenario, dargestellt in Kapitel 3.4.2, sollen auf einfachem Weg Funktionen konsolidiert werden. Allerdings ist die Problematik starker Abhängigkeiten innerhalb der BSW im klassischen AUTOSAR-Ansatz noch nicht vollständig gelöst [152].

Zur örtlichen Abschottung von SWCs stehen OS-Applikationen als logische Container zur Verfügung, welche den Speicherschutz unter Verwendung der MPU konfigurieren. Die gesamte BSW wird dabei meist innerhalb einer geschlossenen OS-Applikation gekapselt, welche Kernel-Rechte besitzt. Somit ist keine feingranulare Aufteilung der BSW möglich. Erste Ansätze einer Verteilung der BSW auf mehrere Kerne sind bereits in Version 4.2.1 verankert [15]. Hierbei bieten Mikrokern-basierte und HV-basierte Systeme spezifischere Möglichkeiten, bestimmten Softwareteilen Kernel-Zugriff zu gewähren. Wiederum muss die BSW vollständig im Kernel-Modus betrieben werden.

Ähnlich zur Lösungsvariante eines HVs stellt ein Mikrokern eine kleine TCB innerhalb eines gemeinsamen AUTOSAR Systems dar. Er trennt somit die Kernfunktionalitäten des OS vom restlichen System ab und ermöglicht dadurch eine gezielte Vergabe von User bzw. Kernel-Zugriffsrechten. Allerdings ermöglicht er keine vollständige Trennung von Abhängigkeiten zwischen einzelnen SWCs und der BSW. Somit müssen relevante BSW Module zusätzlich nach dem geforderten ASIL qualifiziert wer-

den. Um eine Trennung anderweitig zu ermöglichen, könnten benötigte BSW-Dienste parallel als ComplexDriver integriert werden. Diese stellen somit sicherheitsrelevanten Funktionen die benötigten Dienste zur Verfügung.

Aufgrund der zusätzlichen Softwareschicht zur Partitionierung bzw. vollständigen Abschottung von Funktionalitäten ist die HV-Lösung führend. Werden bereits existierende Systeme konsolidiert, müssen bestehende Sicherheitsmechanismen (softwareseitig) nicht weiter überarbeitet und neu ausgelegt werden. Würden SWCs auf einer gemeinsamen BSW integriert werden, bestehen nach wie vor Unklarheiten, wie mit der Vielzahl an Abhängigkeiten zwischen der Applikationsschicht und Teilen der BSW umgegangen werden soll (siehe Kapitel 4). Sicherheitsrelevante und nichtsicherheitsrelevante Systeme werden innerhalb von VMs gekapselt, wodurch eine Fehlerfortpflanzung zwischen VMs vermieden werden kann (Stärkung der Anforderung zur Rückwirkungsfreiheit zwischen SWCs). Der HV fungiert dabei als minimale Isolationsschicht innerhalb des Gesamtsystems und stellt die Trennung zwischen VMs sicher. Allerdings muss dieser nach dem höchst integrierten ASIL qualifiziert werden.

6.7.5 Skalierbarkeit

Die Bewertung der Skalierbarkeit umfasst einerseits die Gesamtkosten des Systems und andererseits den Aufwand zur Migration virtualisierter Steuergeräte. Somit hängt die Aussage der Skalierbarkeit der dargelegten Lösungsvarianten stark mit bereits genannten NFEs zusammen. Die jeweiligen Kosten sind bereits in Kapitel 6.6.5 aufgeführt und benötigte Integrationsaufwände in Kapitel 6.6.8 zusammengefasst und abgewogen.

Werden mehrere AUTOSAR Systeme auf einer gemeinsamen BSW zusammengeführt, entstehen zusätzliche Konfigurationsaufwände innerhalb der BSW. Diese unterscheiden sich unter anderem, ob auf ein identisches Grundsystem migriert wird oder eine vollständig neue BSW zum Einsatz kommt. Grundsätzlich gilt die Annahme, dass alle SWCs von einer zentralen BSW aus verwaltet werden [15]. Im Fall eines Mehrkernsystems entstehen hierbei starke Abhängigkeiten zum Hauptkern, welcher für die Verwaltung der Systemressourcen verantwortlich ist. Es existieren bereits Konzepte zur Funktions- und Lastverteilung der BSW [15], welche allerdings zum Erstellungszeitpunkt dieser Arbeit noch nicht verfügbar sind. Aktuell wird der gesamte Informationsaustausch auf diesem Wege kanalisiert. Dies umgeht die Problematik gleichzeitiger Zugriffe auf gemeinsam verwendete Hardwareressourcen, da diese exklusiv von einem Rechenkern aus angesprochen werden.

Der Mikrokernansatz wird aus Kostensicht zur Funktionskonsolidierung innerhalb einer gemeinsamen Hardwareplattform als gleichwertig zu einem klassischen AUTOSAR System angesehen. Er verfügt die identischen Mechanismen eines klassischen AUTOSAR Mehrkernsystems und ergänzt den Funktionsumfang des OS lediglich um die Funktionalität eines Mikrokerns. Allerdings ist zum Erstellungszeitpunkt der Arbeit noch keine Lösungsvariante im Rahmen eines Mehrkernsystems verfügbar um eine detaillierte Analyse durchführen zu können.

Virtualisierte Systeme und deren VMs sind abgeschlossene Cluster und können auch auf Mehrkernsystemen mit einer Vielzahl von Rechenkernen integriert werden. Zukünftige Hardwarecontroller tendieren zu einer stetig wachsenden Anzahl von Rechenkernen. Diese können frei zu VMs zugeteilt werden. Sollten Nachrichten zwischen Rechenkernen ausgetauscht werden, unterliegen diese Mechanismen den gleichen Restriktionen wie innerhalb eines klassischen AUTOSAR Systems (z.B. falls mehrere Kerne einer VM zugeteilt werden könnten). Begründet durch den nicht-intrusiven Integrationsansatz und der Kapselung von Funktionen innerhalb von VMs können diese nicht feiner auf unterschiedliche Rechenkerne verteilt werden, um eine optimale Lastenverteilung zu erzielen. Allerdings bietet eine zusätzliche Softwareschicht einen weiteren Freiheitsgrad, um konsolidierte Systeme (siehe Szenario 3 in Kapitel 3.4.2) partitionieren zu können.

6.8 Zusammenfassung des Kapitels

Im Rahmen dieses Kapitels wurden Kosten paravirtualisierter Fahrzeugfunktionen analysiert. Dabei wurden sowohl die Architekturprinzipien eines monolithischen, Mikrokern-basierten als auch Hypervisor-basierten Systems gegenübergestellt. Da virtualisierte Echtzeitsysteme noch keinen Einsatz innerhalb aktueller eingebetteter Fahrzeugprojekte gefunden haben, wurde eine weiterführende Kostenanalyse unter Verwendung eines eingebetteten HVs durchgeführt. Dabei wurden die nichtfunktionalen Eigenschaften Wartbarkeit, Skalierbarkeit, Effizienz, Portabilität und funktionale Sicherheit näher beleuchtet.

Als Ergebnis der Analysen dieses Kapitels resultiert, dass der Betrieb von virtualisierten Fahrzeugsoftwaresystemen, unter Verwendung des HVs von ETAS Ltd., grundsätzlich möglich ist. Messungen haben gezeigt, dass hierfür erhebliche zeitliche Mehraufwände in Kauf genommen werden müssen. Unter anderem wird die Verarbeitungszeit für CAT1 Interrupts um 920% bzw. für CAT2 Interrupts um 540% zum eigentlichen Grundwert erhöht (siehe Kapitel 6.6.5). Dies stellt ein Vielfaches des zeitlichen Aufwands im Vergleich zu nativen Systemen dar. Im Gegensatz zur klassischen AUTOSAR-Vorgehensweise werden Funktionen nicht auf einer gemeinsamen BSW zusammengefasst.

In virtualisierten Plattformen werden Softwaresysteme an notwendigen Stellen paravirtualisiert und weitgehend unberührt in ihrer Grundkonfiguration auf eine gemeinsame Hardwareplattform überführt. Die dargestellten Messergebnisse stellen dabei lediglich eine Momentaufnahme des verwendeten HVs und der eingesetzten Hardware dar. Somit hängen diese Ergebnisse stark vom jeweiligen Mikrocontroller und dessen Virtualisierungsunterstützungen ab. Besitzen zukünftige Controller Erweiterungen hinsichtlich virtualisierter Systeme, können die erzielten Messergebnisse verbessert werden.

Eine zusätzliche HV-Softwareschicht bietet somit erweiterte Partitionierungsmöglichkeiten, um vollständige AUTOSAR Systeme voneinander abzuschotten. Dieser Grundsatz ist nicht auf das bestehende Szenario beschränkt und kann auf andere Virtualisierungslösungen übertragen werden. Virtualisierte ECUs werden innerhalb

von VMs gekapselt, um eine systemweite Fehlerausbreitung zu verhindern (siehe Kapitel 6.11). Werden mehrere AUTOSAR Systeme konsolidiert, ermöglicht es ein HV einzelne Systemcluster unabhängig voneinander zu betreiben.

Im Rahmen dieser Analysen wurden folgende Punkte als Schwachstellen identifiziert und bieten Verbesserungspotential für zukünftige Softwareintegrationen. Diese können durch den Anwender der bestehenden HV-Lösung mit der vorgegebenen Hardware softwaretechnisch gelöst werden und erfordern keine Modifikation zur Verfügung gestellter Komponenten:

1. Die Kommunikationskanäle bzw. Schnittstellen zwischen virtualisierten Steuergeräten sind nicht vollständig spezifiziert.

2. Es existiert kein einheitliches Verfahren zur Ansteuerung geteilter Hardwareressourcen, insbesondere für I/O-Kommunikationscontroller.

3. Es bedarf einer Methode zur Initialisierung der Hardware für sicherheitsrelevante Steuergeräte während der Aufstartphase einer ECU (siehe Kapitel 6.6.8).

7 Kommunikationskanäle für virtuelle Steuergeräte

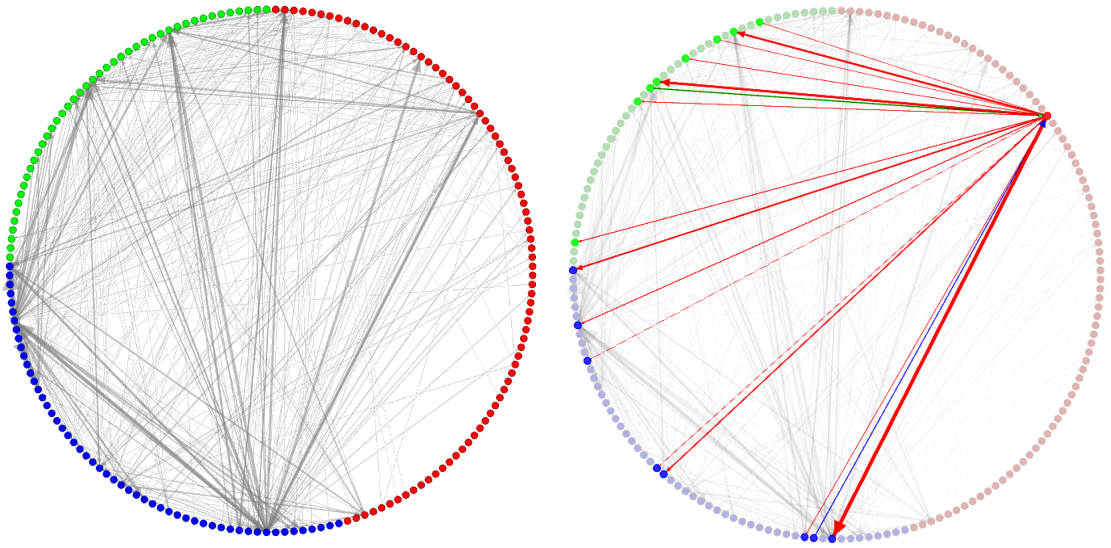
Die Beurteilung des Laufzeitbedarfs, sowohl innerhalb eingebetteter VMs als auch in Zusammenspiel mit dem zugehörigen HV, wurde bereits in Kapitel 6 aufgeführt. Ziel dieses Kapitels ist hingegen die Erstellung und Analyse eines Konzeptes zur Verbesserung des ECU-internen Kommunikationsaustauschs zwischen VMs. Dies umfasst gleichzeitig auch die Steuerung paralleler Zugriffe auf Peripheriecontroller für den ECU-externen Signalaustausch. Hierfür soll ein gemeinsamer Kommunikationskanal zwischen VMs implementiert und evaluiert werden. Weiterhin soll innerhalb dieses Kapitels ein Partitionierungskonzept für Treibermodule des AUTOSAR MCALs zur gemeinsamen Ansteuerung von Kommunikationscontrollern geschaffen und analysiert werden. Hierbei wird der nicht-intrusive Integrationsansatz zur Konsolidierung von Software verfolgt (siehe Kapitel 6.1). Als Designrichtlinie gilt es, die Konfiguration und den Quellcode migrierter Systeme so wenig wie möglich zu verändern. Darüber hinaus soll die Echtzeitfähigkeit virtualisierter Systeme erhalten bleiben. Dadurch sollen die Integrations- und Qualifizierungsaufwände sicherheitsrelevanter Fahrzeugsysteme erleichtert werden.

7.1 Herausforderung und Problemstellung

Die Steuergeräte eines Fahrzeugs weisen meist einen **hohen Funktionsvernetzungsgrad** auf (siehe Abbildung B.5). Abhängigkeiten von einzelnen Applikationen und Komponenten wirken somit meist über Steuergerätegrenzen hinaus. Abbildung 7.1 zeigt hierzu die Abhängigkeiten von farblich markierten Softwarekomponenten auf Busebene, die auf drei typischen Integrationssteuergeräten verteilt sind.

Feldbusse müssen nach erfolgter Systemvirtualisierung durch **virtualisierte Bussysteme** nachgebildet werden und sollen ein identisches Verhalten gegenüber integrierten VMs aufweisen. Daher ist ein generisches Kommunikationsprotokoll zwischen VMs notwendig. Gleichzeitig sollen hierfür notwendige Rechenkapazitäten das eigentliche System innerhalb einer VM nicht beeinflussen.

Für den internen Informationsaustausch zwischen VMs sind bisher **keine standardisierten Schnittstellen** festgelegt. Somit kann nicht sichergestellt werden, in welchem Format Nachrichten an den HV zur Übertragung übergeben werden. Der Kommunikationskanal muss somit unterschiedliche Nachrichtenformate verarbeiten können.



(a) Darstellung aller Abhängigkeiten integrierter Softwarekomponenten.

(b) Vernetzungsgrad einer Softwarekomponente durch benachbarte Systeme.

Abbildung 7.1: Fahrzeugbus-übergreifender Vernetzungsgrad zwischen farbig gekennzeichneten Softwarekomponenten von drei Integrationssteuergeräten [126].

Innerhalb einer Integrationsplattform erfolgt zwischen SWCs ein gleichzeitiger Zugriff auf **gemeinsame Kommunikationscontroller**. Der Umgang mit konkurrierenden Zugriffen wird von den meisten Mikrocontrollern für Fahrzeugsysteme nicht unterstützt. Zur Regelung dieser Zugriffe bedarf es entweder einer regelnden Kontrollinstanz, oder Hardwaretreiber müssen dementsprechend angepasst und konfiguriert werden.

Soll eine Vielzahl von virtualisierten Steuergeräten mit **unterschiedlichen Betriebssystemen und Softwarestandards** auf einer Plattform konsolidiert werden, ist eine Konfigurationsabstimmung der Hardwaretreiber ohne zusätzliche Aufwände nicht möglich. Daher sind neue Konzepte notwendig, um Hardwaretreiber auch zwischen heterogenen Systemen aufteilen zu können.

Eine besondere Erschwernis stellt die **Vielzahl indirekter Abhängigkeiten zwischen sicherheitsrelevanten Systemen** dar. Sollen SWCs auf einer Hardware-Plattform konsolidiert werden, sind sie nicht mehr durch einen Fahrzeugbus physisch voneinander getrennt. In Kapitel 3.3.3 wurde hierzu die Problemstellung zur Vererbung von Sicherheitsleveln erläutert. Die Unterbindung der Vererbung von Sicherheitseinstufungen ist unter direkter Verwendung gemeinsamer Hardwareressourcen und -treiber erschwert.

7.2 Verwandte Methoden und Techniken

Hinsichtlich Performanz birgt Paravirtualisierung Vorteile gegenüber Vollvirtualisierung und wird bereits erfolgreich in Rechenzentren eingesetzt (siehe Tabelle 6.1).

Bereits genannte Vertreter sind Xen [34] oder auch VMware. Diese finden aufgrund nicht hinreichender Echtzeiteigenschaften bisher in eingebetteten Fahrzeugsoftwaresystemen kaum verbreitete Verwendung.

Chiphersteller, darunter Intel oder AMD, erweitern ihre Prozessoren mit fortgeschrittenen Virtualisierungsfeatures wie Intel VT oder AMD-V. Diese Technologien werden bereits in Rechenzentren verwendet, um eine Konsolidierung von unterschiedlichen Systemen auf einem gemeinsamen Serversystem zu ermöglichen. Zur Unterstützung von I/O-Virtualisierung existieren bereits Erweiterungen wie Intel VT-c (*Virtualization Technology for Connectivity*) [87].

Aktuelle Forschungsergebnisse hardwareseitiger Virtualisierungsunterstützung einzelner Controller (*engl. self-virtualized hardware*) eingebetteter Systeme wurden von Herber et al. [79] und Rauchfuss et al. [119] veröffentlicht (für CAN und Ethernet) . Hierbei sind weder Softwareemulation der integrierten Hardware noch Nachrichtenarbitrierungskonzepte notwendig, da diese Vorgänge Controller-intern gelöst sind.

Für sicherheitsrelevante Systeme wurde von Herber et al. [77] ein hardwaregestütztes Arbitrierungsverfahren vorgeschlagen. Eine praktische Umsetzung zur softwareseitigen Emulation der CAN-Controller auf dem Infineon AURIX TriCore für Automobilsysteme wird von Schneider et al. [133] aufgezeigt. Eine Methode, einer hardwaregestützten Mikrocontrollererweiterung in Verbindung mit dem AVB-Protokoll wird in [78] aufgeführt.

Die in dieser Arbeit verwendeten Ansätze und Architekturen leiten sich vom Ansatz des Xen HVs ab [34]. Xen integriert zur Verwaltung der Hardwaretreiber eines Steuergerätes eine spezielle VM namens *Dom0*. Diese verfügt sowohl über Front- als auch Backend-Treiber (genannt *Netfront* und *Netback*), um Informationen zwischen Gastsystemen, den *DomU's* und der verwaltenden Dom0 auszutauschen. Somit übernimmt die oben genannte Dom0 das Routing von Nachrichten und integriert alle Treiber, die von den einzelnen DomUs benötigt werden. Innerhalb der Dom0 werden Ein- bzw. Ausgangssignale konsolidiert und zwischen den Gastsystemen aufgeteilt.

7.3 Anbindung paravirtualisierter Hardwaretreiber

In dieser Arbeit werden alle Gastsysteme und deren Hardwaretreiber zur Integration innerhalb einer virtualisierten Umgebung paravirtualisiert. Dieser Entschluss ermöglicht sowohl zusätzliche Geschwindigkeitsvorteile gegenüber vollvirtualisierten Systemen, als auch softwarebasierte Ergänzungen bzgl. fehlender Virtualisierungserweiterungen der Hardware (siehe Kapitel 6.6). Es existieren dabei unterschiedliche Integrationsansätze, um den Zugriff einzelner Hardwareressourcen durch VMs zu gewähren. Im Folgenden werden die unterschiedlichen Ansätze zur Integration des AUTOSAR MCALs, zur Handhabung von gleichzeitigem Zugriff auf gemeinsame Hardwareressourcen analysiert.

7.3.1 Realisierung des Zugriffsschutzes von Hardwareressourcen

Für den Infineon TriCore AURIX Mikrocontroller existieren drei unterschiedliche Konfigurationen zur Einschränkung von Zugriffsrechten. Im Kernel-Modus (bzw. SU-Modus) existieren keine Restriktionen, um auf Speicher und Peripherie-Controller zugreifen zu können. Für den User-Modus existieren zwei Unterscheidungen: der User-0 Modus schränkt den Zugriff auf Peripherie-Controller vollkommen ein; User-1 Modus gewährt diesen (siehe Kapitel 2.5). Für beide Konfigurationen wird der Zugriff nur auf nicht-geschützte Register gewährt. Jeder Registerzugriff, entweder in User-0 oder User-1 Modus, muss entweder von der ISA erlaubt sein, oder muss im Kontext des HVs im Kernel-Modus ausgeführt werden. Diese Zugriffe werden in jedem Fall durch den HV verifiziert. Sind diese nicht erlaubt, werden solche Zugriffe von der MPU gefiltert. Um den Zugriff auf Kommunikationscontroller beurteilen zu können, werden im Rahmen dieser Arbeit unterschiedliche Integrationsansätze durch individuelle Testaufbauten auf dem Infineon TriCore AURIX Mikrocontroller zur weiteren Analyse implementiert. Hierzu werden folgende drei Anwendungsfälle identifiziert:

1. **Direct Device Assignment**: Die exklusive Anbindung von Hardwaretreibern an VMs ist eine Technik für virtualisierte Server Systeme bzw. Cluster in Rechenzentren, um Rechenaufwände durch den HV einzusparen. Kommunikationssysteme für CAN oder Ethernet müssen nicht softwareseitig aufgeteilt werden. Sie werden den VMs somit exklusiv zugewiesen. Für den TriCore AURIX Mikrocontroller sollten für VMs User-1 Zugriffsrechte eingeräumt sein. Hierfür müssen Zugriffe auf Kommunikationscontroller nicht in den MPU-Einstellungen hinterlegt sein, da der Zugriff grundsätzlich erlaubt wird. Wird eine feinere Kontrolle im System gefordert, müssen im User-0 Modus alle Adressbereiche innerhalb der kernlokalen MPU-Konfiguration hinterlegt werden. Natürlich wird dieses Vorgehen durch die Anzahl verbleibender MPU-Adressbereiche innerhalb der PRSs-Register limitiert. Weiterhin müssen ausreichend Hardwareressourcen zur Verfügung gestellt werden, welche exklusiv an VMs zugewiesen werden können.

   Beispiel: Der CAN-Controller des AURIX Mikrocontrollers besitzt vier CAN-Nodes. Diese können individuell an VMs und an unterschiedliche Feldbusse zugewiesen werden (siehe Kapitel 2.5). Diese Konfiguration wird innerhalb der Initialisierungsphase festgelegt. Identisch zum nativen System erfolgt die Adressierung der Nachricht und der Zuweisung deren Nutzdaten direkt innerhalb der Register des Controllers.

   Aufgrund des direkten Zugriffs auf Register des Mikrocontrollers ist das Laufzeitverhalten nahezu identisch im Vergleich zum nativen Systemaufbau. Dies ist durch die exklusive Zuweisung von CAN-Nodes und Rechenkernen begründet.

   Allerdings besteht weiterhin das Problem der Initialisierung bzw. korrekten, systemweiten Konfiguration des CAN-Controllers und dessen Nodes. Diese Komponenten werden einmalig beim Aufstarten des Systems nacheinander initialisiert.

Somit benötigt eine Instanz innerhalb der ECU das Wissen über die Anforderungen aller VMs (z.B. Busgeschwindigkeit).

2. **Paravirtualisierung von exklusiven Hardwareressourcen**: Sind die Adressbereiche einer MPU nicht ausreichend, um den Zugriff auf Kommunikationscontroller zu gewähren, müssen relevante Aufrufe paravirtualisiert werden. In diesem Fall muss jeder Aufruf einer VM durch einen Hypercall versehen werden, um eine potentielle Modifikation eines Registers im Kontext des HVs auszuführen. Besitzt eine VM lediglich User-0 Zugriffsberechtigung, ist dieser Schritt ohnehin notwendig. Aufgrund paravirtualisierter Codeanteile und die dadurch benötigten Hypercalls erhöht sich das Trap-Aufkommen im System. Somit verursacht dieses Vorgehen einen zusätzlichen Bedarf an Rechenzeit und verschlechtert die Systemperformanz.

   Beispiel: Der AURIX Mikrocontroller besitzt begrenzte Adressbereiche für Daten innerhalb jeder kernlokalen MPU-Konfiguration, exklusiv für eine VM (16 Data Protection Ranges (DPRs)) und Code (8 Code Protection Ranges (CPRs)). Im Fall von RTA-HV verbleiben unter Verwendung der Grundeinstellungen 6 Daten- und 5 Code-Adressbereiche und stehen zur freien Verfügung innerhalb einer VM. Verbleibende Bereiche sind bereits vom HV zum Schutz der unterschiedlichen Stacks (CSA, User Stack, Interrupt Stack) reserviert.

   Die meisten Anpassungen hinsichtlich Paravirtualisierung sind im Startup-Code der Treiber vorzunehmen. Wird der Zugriff auf Peripherieregister durch User-1 Modus gewährleistet, sind innerhalb der Treibermodule nur wenige Anpassungen am Code notwendig.

   Ein Nachteil stellt die Initialisierung der Treibermodule während der Aufstartphase des Steuergerätes dar. Diese darf ausschließlich von einer Instanz vorgenommen werden. Zu migrierende ECUs führen diesen Initialisierungsschritt vorerst individuell aus. Innerhalb paravirtualisierter Umgebungen darf dieser Vorgang für virtualisierte Steuergeräte nicht erfolgen. Andernfalls würde die Konfiguration der Hardware während jedem Konfigurationsvorgang erneut überschrieben werden.

3. **Emulation von gemeinsamen Hardwareressourcen**: In diesem Fall benötigt mehr als eine VM Zugang zu einer bestimmten Hardwareressource. Als einzige regulierende Instanz im System muss der HV diesen Zugriff kontrollieren und arbitrieren. Hierfür kann ein VDE zur Kontrolle der Anfragen auf gemeinsamen Hardwareressourcen, ausgehend von VMs, als Router dienen. Der zeitliche Aufwand ist vergleichbar mit Anwendungsfall 2 zur Paravirtualisierung von Hardwareressourcen. Jedoch müssen innerhalb der Implementierung gleichzeitige Registerzugriffe berücksichtigt werden. Dies verursacht zusätzliche zeitliche Aufwände durch Schutzkonzepte, wie die Einführung von Spinlocks (blockierend).

   Beispiel: Der AURIX Mikrocontroller besitzt nur einen Ethernet Controller. Sollen mehrere VMs auf diese Peripherie zugreifen, muss diese Hardwareressource

emuliert werden. Alle eingehende bzw. ausgehende Nachrichten müssen zwischen VMs geroutet werden.

Der Vorteil dieser Methode ist, dass VMs kein Wissen über vorhandene Hardwareressourcen besitzen müssen. Somit werden die Nutzdaten einer Nachricht direkt an den HV übergeben. Die Adressierung sowie die Befüllung relevanter Register des Controllers werden über den HV gesteuert und kontrolliert.

Als Nachteil daraus resultiert der zusätzliche Implementierungsaufwand für einen notwendigen VDE, der relevante Hardwareressourcen emuliert. Weiterhin muss der vorhandene MCAL Treiber innerhalb der VM adaptiert werden, sodass Registerzugriffe auf den betreffenden Mikrocontroller vermieden und Nutzdaten direkt an den virtualisierten Controller übergeben werden.

7.3.2 Integrationsansätze von AUTOSAR Treibermodulen

Zur Ansteuerung eines Peripheriecontrollers innerhalb des AURIX Mikrocontrollers sind Hardwaretreiber notwendig. Typischerweise wird der direkte Hardwarezugriff von AUTOSAR SWCs unterbunden. Zugriffe werden durch mehrere Schichten der BSW geleitet. Hardwaretreiber sind innerhalb des MCALs implementiert (siehe Kapitel 2.3). In einem hochintegrierten System müssen unter Umständen Hardwareressourcen zwischen mehreren VMs geteilt werden. Diese Tatsache stellt eine Herausforderung für Systeme dar, in deren VMs mit unterschiedlichem ASIL integriert sind. Hierbei kann ein unkontrollierter Zugriff bzw. die Aufteilung von Hardwareressourcen zu Fehlern, wie Verklemmungen, oder zu Prioritätsproblemen führen [89, 133]. Besitzt diese VM die Kontrolle über relevante Treiber bzw. Hardwareressourcen, kann ein störungsfreier Betrieb für verbleibende VMs nicht mehr garantiert werden. Zur Verlagerung der MCAL-Schicht der AUTOSAR BSW werden folgende drei Anwendungsfälle identifiziert und zu Demonstrationszwecken implementiert (siehe Nummerierungen in Abbildung 7.2):

1. **Treibermodule werden innerhalb der VMs integriert**: Alle Treiber sind innerhalb der VMs integriert und werden somit vollständig im User-Modus ausgeführt. Entweder besitzt eine VM direkten Zugriff auf Peripheriecontroller (durch User-1 Modus) oder alle Registerzugriffe müssen (begründet durch User-0 Modus) im Kontext des HVs ausgeführt werden. In jedem Fall benötigt die jeweilige VM exklusiven Zugriff auf betreffende Peripheriecontroller (DDA). Alle Register, welche alleinig im Kernel-Modus verändert werden, müssen somit im Kontext des HVs angesprochen werden. Zum Aufruf des HVs werden innerhalb des verwendeten paravirtualisierten Ansatzes Hypercalls verwendet. Um lesenden bzw. schreiben Registerzugriff zu ermöglichen, wird die Instruktion MFCR bzw. MTCR aufgerufen und im HV emuliert.

   Vorteil dieses Ansatzes ist der geringe Änderungsaufwand für migrierte Systeme. Lediglich relevante Registerzugriffe, welche Kernel-Berechtigung erfordern, müssen durch Hypercalls bzw. freigegebene Instruktionen (wie MTCR) ausgeführt

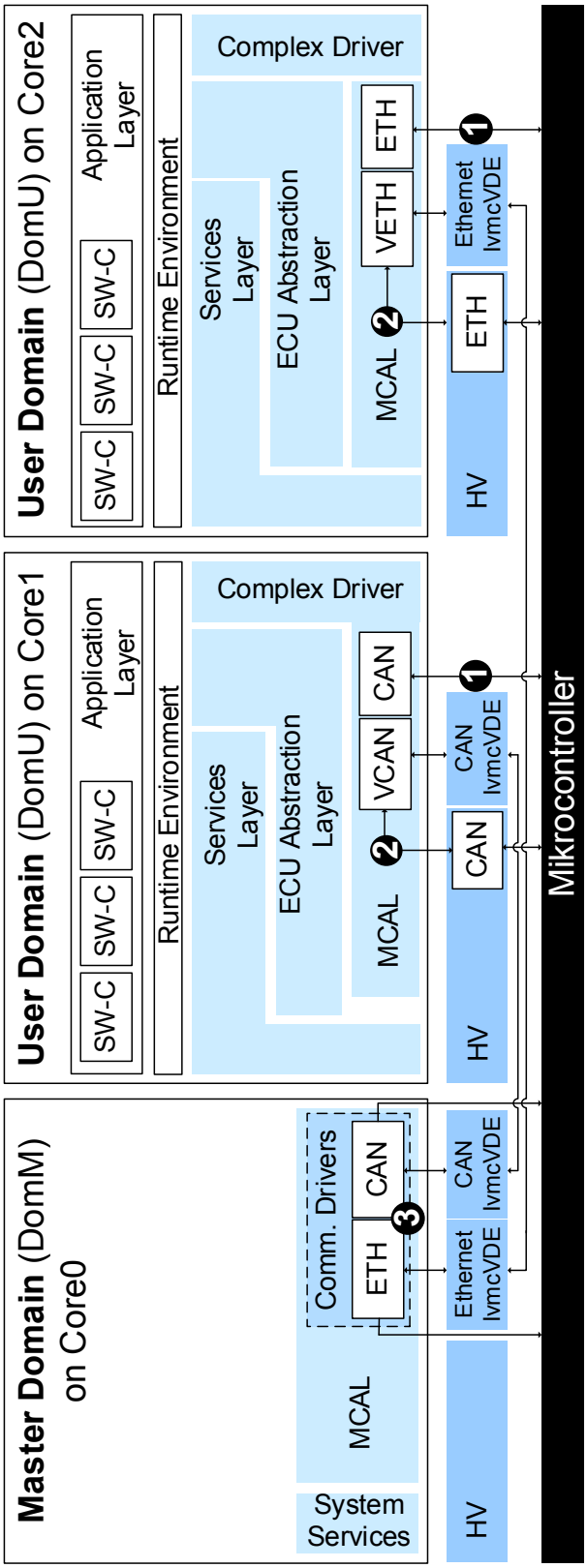


Abbildung 7.2: Ansätze zur Verlagerung der AUTOSAR Treiberschicht [126].

werden. Somit ist das Gesamtsystem durch potentiell fehlerhafte Treibermodule geschützt. Falls ausreichend Hardwareressourcen verfügbar sind, können diese direkt an VMs zugewiesen werden (DDA). Sind einzelne VMs fehlerhaft, werden verbleibende Gastsysteme in ihrer Ausführung nicht gestört.

Im Gegensatz dazu steht, dass Hardwareressourcen in eingebetteten Systemen eingeschränkt zur Verfügung stehen. Um Rückwirkungsfreiheit sicherstellen und zeitliche Einbußen vermeiden zu können, müssen alle Peripheriecontroller physikalisch getrennt auf einem Chip integriert sein. Andernfalls sind Hardwareressourcen vom HV zu emulieren und zwischen den VMs aufzuteilen. Sollen trotzdem exklusive Hardwareressourcen von mehreren VMs verwendet werden, sind alle Anfragen zwischen VMs auszutauschen. Signale müssen somit unter Verwendung von HV-Mechanismen an die VM geleitet werden, welche den betreffenden Treiber integriert. Dort werden die Signale empfangen und darauffolgend an die Hardware übertragen.

2. **Treibermodule werden im HV integriert**: Der gesamte AUTOSAR MCAL und dessen Treibermodule werden im HV integriert. Alle Treiber werden somit im Kernel-Modus ausgeführt und erhalten Vollzugriff auf Hardwareressourcen. Die Ansteuerung erfolgt über mehrere VDEs. Jeglicher Zugriff auf Hardwaretreiber und -register wird vom HV verwaltet und gesteuert.

   Der Vorteil dieser Methode ist, dass Hardwareressourcen zentral angesprochen werden und VMs gleichberechtigten Zugriff erhalten. Weiterhin besteht kein Bedarf betreffende Treiber innerhalb des HVs zu paravirtualisieren, da diese bereits Vollzugriff auf Hardwareregister besitzen. Somit besteht der einzige zeitliche Mehraufwand darin, Serviceanfragen den verlagerten AUTOSAR MCAL an den HV weiterzuleiten. Der verbleibende Source Code kann in nativer Geschwindigkeit ausgeführt werden. Bei Neustart oder gar Entfall einer VM können verbleibende VMs weiterhin auf global zugängliche Hardwaretreiber zugreifen. Eine Anforderung der AUTOSAR BSW ist die Möglichkeit, das Softwaresystem flexibel auf neue Hardwarecontroller portieren zu können [21]. Hierfür wird der bisher integrierte MCAL durch eine kompatible Variante ausgetauscht. Dies garantiert eine Entflechtung des Grundsystems von dem betreffenden Treibermodul, zur einfachen Verlagerung in den HV.

   Problematisch hingegen ist der Umgang mit fehlerhaften Treibermodulen, die ein unkontrollierbares Verhalten im System hervorrufen könnten. Innerhalb des HV wird jeglicher Source Code im Kernel-Modus ausgeführt und besitzt dadurch Vollzugriff auf das System. Somit kann das Gesamtsystem und der HV selbst beeinträchtigt werden. Zudem vergrößert eine Treiberverlagerung in den HV die TCB, wodurch Qualifizierungskosten ansteigen. Weiterhin sollte auf Sperrmechanismen innerhalb des HVs weitestgehend verzichtet werden, um die dortige Verweildauer so gering wie möglich zu gestalten. Diese werden jedoch zur Befüllung von Registern und anderen Hardwareressourcen häufig verwendet. Sobald ein Spinlock einer HV-Instanz innerhalb des betreffenden VDEs blockiert, können andere zugreifende VMs den geschützten Bereich nicht betreten. Für alle

VMs und Rechenkerne ist der Quellcode der VDEs eintrittsinvariant. Dadurch kann eine zeitliche Rückwirkungsfreiheit zwischen VMs nicht mehr sichergestellt werden. Grundsätzlich sind Interrupts während der Ausführungszeit des HVs oder innerhalb von Traproutinen ausgeschaltet (siehe Kapitel 6.6.2). Innerhalb dieser Zeitspanne werden keine Interrupts entgegengenommen. Im Fall einer hohen Interruptlast durch externe Eingangssignale werden unter Umständen nicht alle Interrupts angenommen, da relevante Treibermodule im HV ausgeführt werden. Dies gefährdet das korrekte Echtzeitverhalten der ECU.

3. **Treibermodule werden in eine privilegierte VM verlagert**: Dieser Ansatz unterscheidet zwischen klassischen Gastsystemen, bezeichnet durch User Domain (DomU), und einer speziellen VM, genannt Master Domain (DomM). Diese Vorgehensweise ist sehr ähnlich zum Xen HV, welcher Steuerungsaufgaben innerhalb einer privilegierten VM namens *dom0* isoliert [34]. Auch eine DomM integriert Hardwaretreiber im Rahmen einer speziell betrachteten VM. Ausschließlich diese Partition ist für das Routing von Nachrichten zwischen DomUs untereinander und angeschlossener Feldbusse verantwortlich. Sie besitzt exklusiven Zugriff (DDA) auf relevante Hardwareressourcen wie Kommunikationscontroller. Eine DomM wird im User-1 Modus betrieben, um grundsätzlich auf Kommunikationscontroller zugreifen zu können und erhält innerhalb der MPU-Konfiguration Zugriffsrechte auf benötigte Register. Die DomM wird innerhalb einer isolierten VM betrieben und implementiert eine Netzwerkbrücke (*engl. Bridge*), um interne Signale an DomUs oder Feldbusse gezielt weiterleiten zu können. Gleichzeitig werden externe Signale eines Feldbusses an DomUs verteilt. Da der HV bereits im Kernel-Modus betrieben wird, kann es aus Sicherheitsgründen einer DomM nicht erlaubt sein mit identischen Zugriffsrechten vorkonfiguriert zu sein. Der HV muss weiterhin die höchst privilegierteste Instanz im System sein.

   Vorteil dieses Ansatzes ist, dass Treibermodule nicht mit Kernel-Rechten versehen werden und die TCB des HVs nicht erhöhen. Somit reduziert sich die Aufenthaltsdauer im HV, wodurch der Empfang von Interrupts seltener blockiert wird. Anstelle von Kernel-Rechten werden User-1 Berechtigungen vergeben. Das Problem potentiell geteilter Hardware wird dadurch gelöst, dass alle Gastsysteme über eine zentralisierte VM auf Treiberschichten zugreifen. Dadurch können eingehende Anfragen einfacher serialisiert werden. Wird eine DomU bzw. VM aufgrund eines Fehlerfalls neu gestartet, werden verbleibende VMs dabei nicht gestört und können weiterhin unter Verwendung der DomM kommunizieren. Weiterhin kann die Nachrichtenverwaltung rechenintensiver Feldbusse mit hoher Bandbreite (wie FlexRay) auf einen separaten Kern ausgelagert werden. Verbleibende Gastsysteme werden daher lediglich mit notwendigen Informationen versorgt.

   Ein offensichtlicher Nachteil ist im Fall der aktuellen Version von RTA-HV der Verlust eines Rechenkerns bei Einführung einer DomM, da Rechenkerne exklusiv den VMs zugeordnet werden (DDA). Weiterhin muss die gesamte Kommu-

nikation zwischen DomUs zusätzlich über die DomM geroutet werden, wodurch zusätzlicher Zeit- und Rechenaufwand entsteht. Dies gilt auch für externe Signale, welche nun über eine zusätzliche Instanz geleitet werden. Weiterhin müssen Kommunikationssemantiken und Austauschkanäle für dieses Konzept aufwendig implementiert werden.

7.3.3 Beurteilung dargestellter Integrationsansätze

Die Aufteilung von Hardwareressourcen verursacht zeitlichen Aufwand zur Arbitrierung von Frames und den Schutz der Peripheriecontroller gegen nicht genehmigte Registerzugriffe. Im Rahmen der Analyse wurde festgestellt, dass innerhalb sicherheitsrelevanter Systeme die Emulation und Aufteilung von Kommunikationscontrollern von einer zentralen Instanz (wie einem HV) überwacht werden muss, um Rückwirkungsfreiheit gewährleisten zu können. Diese Tatsache ist nicht auf den verwendeten Controller fixiert und auf weitere Plattformen übertragbar.

Weiterhin sei angemerkt, dass sobald Hardwaretreiber im HV oder innerhalb einer DomM integriert sind, die Ausfallsicherheit dieser Partitionen garantiert werden muss. Beide Methoden gefährden zum aktuellen Versionsstand des HVs bei Versagen das Gesamtsystem (*Single Point of Failure*). Der Fehlerfreie bzw. -tolerante Betrieb des HVs bzw. der DomM muss daher durch zusätzliche Sicherungsmechanismen, erweiterte Qualifizierungsmaßnahmen oder durch Redundanz sichergestellt werden.

Sind **Hardwaretreiber innerhalb der VMs** integriert, müssen Controller bzw. Hardwareressourcen **exklusiv den VMs zugewiesen** werden. Diese Methode ist allerdings nur möglich, wenn keine Hardwareressourcen geteilt werden müssen. Werden kleinere Steuergeräte auf einer gemeinsamen Hardwareplattform konsolidiert, kann nicht angenommen werden, dass alle Hardwareressourcen exklusiv verteilt werden können. Somit ist diese Herangehensweise für weitere Analysen nicht geeignet.

Sind **Treiber im HV** integriert, müssen für VMs **Peripheriecontroller emuliert** werden. Dadurch nimmt der Umfang der TCB des Gesamtsystems zu. Darüber hinaus fallen zusätzliche Rechenaufgaben innerhalb des HVs an. Jeglicher Hardwarezugriff wird daher im HV verarbeitet. Aufgrund von zeitlichen Einschränkungen, den zusätzlichen Rechenaufwänden zur Arbitrierung innerhalb des HVs und der Erweiterung der TCB scheidet diese Methode somit aus.

Im Folgenden wird daher die Verlagerung der **Treiberschicht in eine unabhängige VM** untersucht. Dieser VM werden **Hardwareressourcen exklusiv zugewiesen** (DDA). Wie in Kapitel 7.3.2 beschrieben wird diese priviligierte VM als DomM bezeichnet, um Treiberschichten von verbleibenden VMs zu entkoppeln. Um den Zugriff auf Hardwaretreiber innerhalb der DomM zu gewähren, werden relevante Adressregionen der kernlokalen MPU exklusiv für diesen Zweck freigegeben (siehe Kapitel 7.3.1). Da eine DomM exklusiven Zugriff auf Hardwareressourcen besitzt, besteht kein Grund zur Emulation bzw. Paravirtualisierung einzelner Peripheriecontroller.

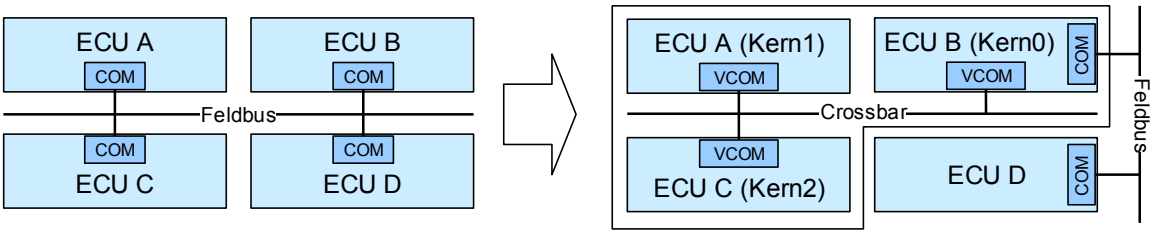


Abbildung 7.3: Konsolidierung einzelner Steuergeräte und Abbildung virtualisierte Busstrukturen [126].

7.4 Informationsaustausch zwischen virtuellen Maschinen

Steuergeräte sind durch Feldbusse miteinander vernetzt und tauschen Informationen aus (siehe Abbildung 7.1). Diese Kommunikationswege müssen Steuergeräte-intern als virtualisierte Bussysteme abgebildet werden (siehe Abbildung 7.3). Dabei soll das bisherige Kommunikationsverhalten nicht verändert oder gar gestört werden. Somit müssen nachgebildete Kommunikationswege mindestens eine identische Übertragungsgeschwindigkeit (Latenz) bzw. identisches Durchsatzvolumen aufweisen, um Kommunikationsengpässe zu vermeiden.

7.4.1 Realisierung eines generischen Kommunikationskanals

Um den Informationsaustausch zwischen VMs und Kernen zu ermöglichen, wird ein spezieller VDE implementiert, der einen generischen Kommunikationskanal für den Datenaustausch bereitstellt [113, 126]. Jeder Austausch zwischen VMs wird vom HV kontrolliert und ankommende Anfragen an den hierfür entwickelten Inter-Virtual-Machine-Communication VDE (IvmcVDE) weitergeleitet (siehe Abbildung 7.4). Der Nachrichtenaustausch erfolgt somit ausschließlich im Kontext des HVs. Dabei soll die Datenablage innerhalb eines gemeinsamen Speicherbereichs erfolgen. Im AURIX Mikrocontroller wird globaler Speicher von der LMU bereitgestellt.

Es sind zwei Herangehensweisen für den Informationsaustausch implementiert. Um Informationen zwischen VMs auszutauschen, sind entweder Polling oder Interrupt-getriebene Kommunikationsmechanismen verfügbar. Beide Strategien implementieren einen Kommunikationskanal zwischen VMs. Alle Austauschdaten werden ausschließlich unter Verwendung von spezifizierten HV-Mechanismen übertragen.

Unter Verwendung von Polling-Mechanismen, muss eine VM neue Informationen innerhalb des globalen Datenspeichers, der LMU des AURIX Mikrocontrollers detektieren. Hierfür sind keine Interrupts oder Traps notwendig. Sind neue Informationen einer Empfänger-VM innerhalb eines Ringpuffers innerhalb der LMU vorhanden, werden diese Daten zyklisch entnommen.

Für ein event-basiertes Verfahren werden Interrupts des HVs verwendet, um VMs zu benachrichtigen. Der Sendekern legt unter Verwendung des IvmcVDEs Daten in einem geschützten Puffer ab und benachrichtigt den relevanten Empfängerkern. Dieser

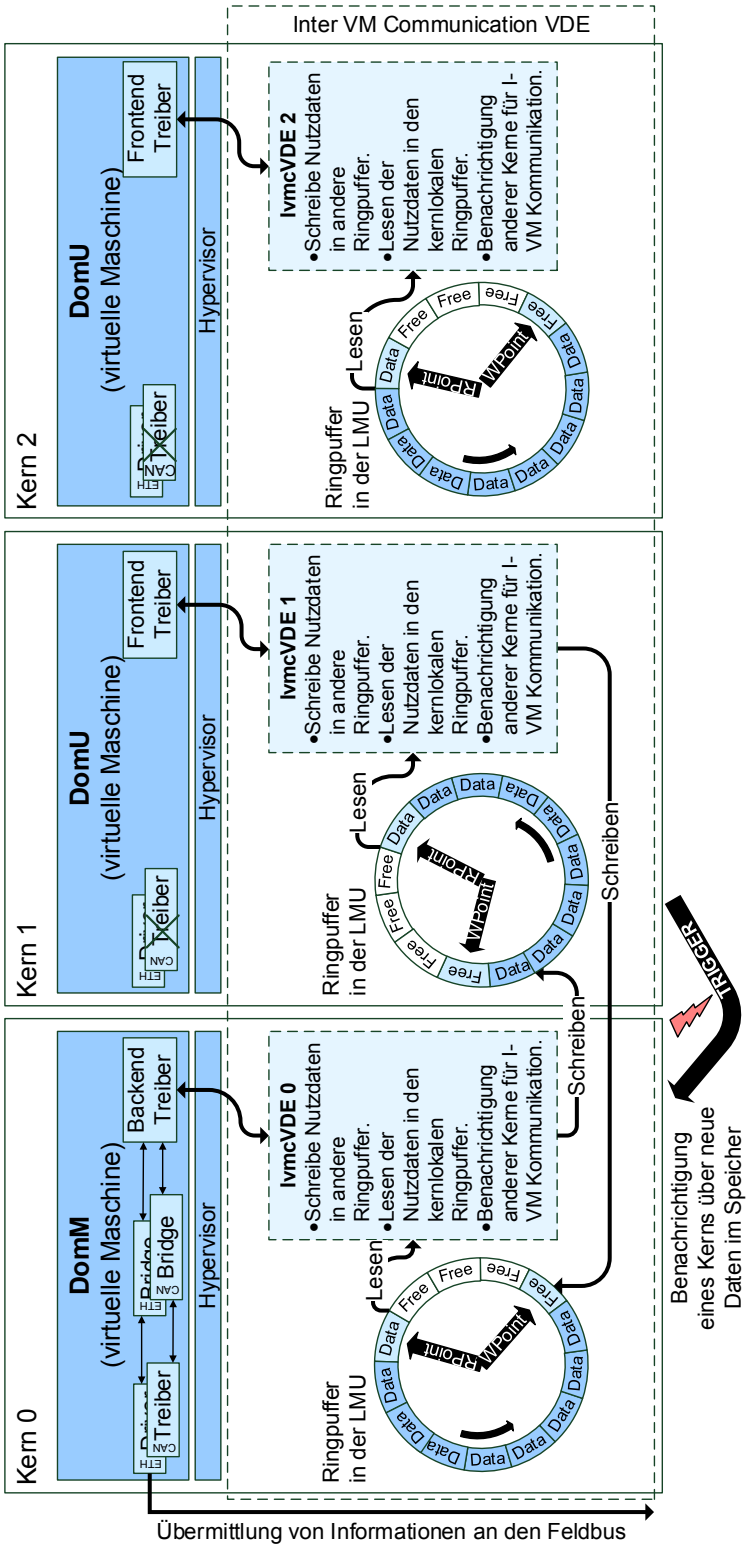


Abbildung 7.4: Kommunikationsmechanismus zur Interaktion zwischen virtuellen Maschinen unter Verwendung von Frontend- bzw. Backend-Treibern [126].

Bit	0	8	16	24	56				
Ivmc	Kom.Mod.	Empfänger	Sender	Länge	Nutzdaten				...
CAN					Nachrichten ID	Länge	Daten		...
ETH					Empf.MAC	Sender MAC	Typ	Daten	...

Tabelle 7.1: Nachrichtenaufbau eines CAN bzw. Ethernet Frames.

ruft nun den IvmcVDE auf, um die neuen Daten aus den Puffern zu laden. Der Quellcode eines VDEs ist eintrittsinvariant und wird somit von mehreren VMs im Kontext des jeweiligen Rechenkerns aufgerufen. Hierbei wird anhand der Kern-ID unterschieden, welcher Rechenkern den Quellcode des VDEs ausführt.

Um blockierende Zugriffe zu vermeiden, wird innerhalb jeder VM ein Ringpuffer implementiert (siehe Abbildung 7.4). Hierbei wird der hardwaregestützte *Circular Buffer Addressing Mode* des TriCore AURIX verwendet [4]. Diese Erweiterung unterstützt atomare Schreib- und Lesezugriffe (*circular pointer shifting*). Wird ein neuer Eintrag einem Ringpuffer hinzugefügt, wird der Schreibzeiger auf das nächste freie Datenfeld verschoben. Werden vom Puffer Daten abgerufen, referenziert der Lesezeiger das nächste beschriebene Datenfeld.

Jeder Puffer wird im globalen Arbeitsspeicher der LMU abgelegt und besitzt eine Größe von 4096 Byte. DomUs schreiben Nachrichten in den Ringpuffer der DomM. Die Verarbeitung und das Weiterleiten der Nachrichten erfolgt im FIFO Prinzip. Dabei sind wiederum zwei Eventkanäle implementiert: Polling (ein Flag im Speicher signalisiert neue Datenstände) und event-basierte Mechanismen unter Verwendung von Interrupts.

Tabelle 7.1 zeigt die grundlegenden Nachrichtenstrukturen innerhalb des Systems. Die Protocol Control Information (PCI) eines jeden Frames ist innerhalb eines zusätzlichen Headers hinterlegt. Der *Kommunikationsmodus* kontrolliert die Nachrichtenverarbeitung innerhalb des IvmcVDEs und unterscheidet dabei zwischen polling- und interrupt-basierte Mechanismen. Das *Empfänger*- und das *Sender*-Feld geben Aufschluss über Ziel- und Quell-VM des Nachrichtenflusses. Um variable Frameläng en für IEEE 802.3 Ethernet zu ermöglichen, wird die Verarbeitung dynamischer Paketlängen unterstützt. Das *Längen*-Feld speichert die im Frame enthaltene Größe der Nutzdaten.

Das *Payload*-Feld enthält das verschachtelte CAN- oder Ethernet Frame. Solange die Kommunikation innerhalb einer sicheren Umgebung stattfindet, in welcher ECC-RAM und Lock-Step Kerne die Korruption übertragener Daten sehr unwahrscheinlich machen, können Prüfsummen für Nutzdaten entfallen. Vor tatsächlichem Versand auf dem Feldbus werden diese vom jeweiligen Treiber oder dem Controller selbst erstellt und der Nachricht beigefügt. Sie müssen also nicht auf dieser Vermittlungsschicht (ECU-intern) zusätzlich hinzugefügt werden.

7.4.2 Schnittstellenbeschreibung zur domänenübergreifenden Interaktion

Zur Paravirtualisierung relevanter MCAL-Kommunikationsmodule werden standardisierte MCAL-Adapter eingeführt, um die Interaktion mit dem IvmcVDE zu erleichtern. Abbildung 7.4 stellt das Verfahren dar, um Daten zwischen einzelnen Domänen übertragen zu können. Diese Arbeit orientiert sich an der Methode von Xen [34] und dem Ansatz von Netfront- bzw. Netback-Treiber, welche auf ähnliche Art und Weise Informationen zwischen VMs vermitteln. Auch dort werden jegliche Hardwaretreiber bzw. Systemanteile mit besonderen Schutzansprüchen in eine *dom0* verlagert. Jede Verbindung zwischen diesen Treibern und den DomU Gastsystemen wird somit vom Xen HV überwacht und kontrolliert.

Die in Abbildung 7.4 dargestellten Frontend-Treiber repräsentieren Adapter, welche Daten der DomUs innerhalb deren Treiberschichten verarbeiten und mit dem zentralen Backend-Treiber der DomM verbinden. Der Aufbau der jeweiligen Adapter zum Empfang und Transfer von Informationen ist nahezu identisch. Im Fall des Backend-Treibers müssen dabei zusätzliche Funktionalitäten für das Routing einzelner Nachrichten hinzugefügt werden. Zum Aufruf des IvmcVDEs ist ein Hypercall innerhalb der jeweiligen VM notwendig (siehe Kapitel 6.6.2).

Der Quelltext 7.1 beschreibt einen Programmablauf im Backend-Treiber zum Empfang von Nachrichten mit fixer Länge innerhalb der DomM. Dieses Vorgehen wird beispielsweise zur Verarbeitung von CAN-Frames verwendet. Daten werden in diesem Quelltextbeispiel durch ein ereignisgesteuertes Verfahren entgegengenommen. Sind neue Datenstände für den Transfer im globalen Speicher der LMU verfügbar, sendet der HV im Kontext der IvmcVDE einen Interrupt an DomM. Dadurch wird innerhalb der VM eine entsprechende Unterbrechungsroutine des Backend-Treibers ausgelöst.

Zuerst wird der Rahmen (*ivmc_message_envelope*) einer Nachricht angelegt, welcher analog zu Tabelle 7.1 die PCI beinhaltet. Weiterhin wird die Struktur der tatsächlichen Nutzdaten der Nachricht angelegt (*ivmc_message*). Das Nutzdatenfeld enthält die eigentlichen Nachrichteninformationen. Der Quelltext 7.1 zeigt dabei einen grundlegenden Programmablauf zum Versand von Nachrichten. Im Fall von Nachrichtenpaketen mit variabler Länge (z.B. Ethernet), muss vorab die Nachrichtengröße ermittelt werden. Wird ein erfolgreicher Nachrichtenempfang verzeichnet, können die Daten innerhalb der VM verarbeitet werden. Der grundlegende Aufbau eines Backend-Treibers ähnelt stark dem Vorgehen zum Versand von Nachrichten, unter Verwendung eines Frontend-Treibers. Der Versand einer Nachricht durch einen Frontend-Treiber erfolgt wiederum durch Verwendung eines Hypercalls, um Informationen an den IvmcVDE zu übermitteln.

7.4.3 Routing von Informationen unter Verwendung einer privilegierten Domäne

Eine DomM benötigt ein Protokoll und Routingschema, um Informationen zwischen DomUs und anderen Feldbussen weiterleiten zu können. Hierfür wird eine Netzwerk-

```
1  void InterruptHandlerPrio6_Partition0(){
2          uint8_t canMsg[8];
3          ivmc_message_envelope me;
4          ivmc_message msg;
5
6          me.payload = &msg;
7          msg.data = &canMsg;
8          me.request = IVMC_REQUEST_READ;
9
10         HV_CallVirtualDevice(IvmcVdeId, &me);
11
12         if (me.request == IVMC_REQUEST_SUCCESS) {
13                 routeMessage(&canMsg);
14         }
15 }
```

Quelltext 7.1: Unterbrechungsroutine zum Empfang von Interrupts des Backend-Treibers innerhalb der DomM.

brücke innerhalb der DomM integriert, welche betreffende Informationen ECU-intern und ECU-extern transferiert. Die implementierte Funktionalität wird in Abbildung 7.4 dargestellt. Diese enthält eine Source Address Table (SAT) für Ethernet und eine Routingtabelle für CAN. Beide beinhalten das Routingprotokoll für eingehende Nachrichten. Eine Netzwerkbrücke arbeitet typischerweise auf der zweiten Schicht des ISO/OSI-Referenzmodells und überbrückt Netzwerkcluster für die Überleitung von Informationen. Die implementierte Netzwerkbrücke ist in einen *Controlling* und einen *Forwarding Part* unterteilt:

Der **Controlling Part** beinhaltet die Routingtabelle (siehe Tabelle 7.2) und die Verwaltung dieser. Werden Ethernet Frames geroutet, existiert hierfür eine SAT, welche die verbundenen MAC-Adressen enthält. MAC-Adressen werden einzeln jedem verbundenen Port zugewiesen [68]. Ports repräsentieren in diesem Fall die jeweilige Ziel-VM. Dieses Verhalten ist identisch verglichen mit einem Netzwerkswitch. Im Fall der CAN-Kommunikation enthält die Routingtabelle CAN-IDs. Daher wird zwischen einem statischen (für CAN) und einem dynamischen (für Ethernet) Tabellenaufbau unterschieden. Die CAN Routingtabelle wird statisch vor der Kompilierung des Softwarestands erzeugt und enthält alle Routinginformationen bereits im Voraus. Die Ethernet SAT wird dynamisch zur Laufzeit erstellt und weist MAC Adressen Port-IDs zu. Letztlich sind beide Tabellen mit einer Lookup-Tabelle vergleichbar. Für jeden Kommunikationskanal ist eine individuelle Routingtabelle notwendig.

Der **Forwarding Part** routet Nachrichten zwischen VMs oder angeschlossenen Feldbussen (siehe Quelltext 7.2). Jede eingehende Nachricht wird entweder durch Polling Mechanismen entgegengenommen, oder durch einen erhaltenen Interrupt innerhalb einer ISR verarbeitet (siehe Quelltext 7.1). Um Nachrichten zu serialisieren und zwischen VMs übertragen zu können, wurde der IvmcVDE implementiert. Um

ID	**Port**($DomM_{Kern0}$)	**Port**($DomU_{Kern1}$)	**Port**($DomU_{Kern2}$)
0x1F	TRUE	FALSE	TRUE
0xA2	FALSE	TRUE	TRUE
...	...	...	...

Tabelle 7.2: Exemplarische Routingtabelle zur Weiterleitung von CAN-Frames.

Nachrichten über einen angeschlossenen Feldbus zu übertragen, beinhaltet die DomM relevante Hardwaretreiber, welche exklusiven Zugriff auf benötigte Hardwarecontroller besitzen (DDA).

Tabelle 7.2 zeigt ein exemplarisches Routingprotokoll für eingehende CAN-Frames. Die Netzwerkbrücke unterscheidet hierbei gemäß der *ID* (dem CAN-Identifier), zu welchem *Port* (der VM und deren Rechenkern) die Nachricht zugeordnet und geroutet werden muss (z.B. Frame ID 0xA2 wird auf Kern 1 und Kern 2 geroutet). Erhält DomM ein eingehendes Signal (siehe Quellcode 7.1), welches auf die DomM selbst verweist, wird diese Nachricht über den angeschlossenen Feldbus übertragen. Dabei werden alle Routinginformationen statisch vor dem Kompilieren konfiguriert.

Werden eingehende Ethernet Frames empfangen, werden diese DomM-intern an den TCP/IP Stack übertragen. Die Netzwerkbrücke unterscheidet dazu aufgrund der *ID* (der MAC-Adresse), zu welchem *Port* (die VM und deren Rechenkern) die Nachricht geroutet werden muss. Ähnlich zum CAN-Ansatz werden mehrfach MAC Adressen zugewiesen (*IDs* auf *Ports*).

Der Quelltext 7.2 zeigt dabei die Routingfunktion, die innerhalb der DomM zum Weiterleiten einer Nachricht aufgerufen wird (siehe Quelltext 7.1). Nach Definition der IVMC-Nachricht wird anhand der Routingtabelle der Empfängerkern aus der Nachrichten PCI ermittelt. Ist dies erfolgt, wird die Nachricht unter Verwendung des IvmcVDEs an die jeweilige VM weitergeleitet.

7.5 Systemanalyse und Beurteilung der Messwerte

In den folgenden Analysen soll der Aufwand zur Paravirtualisierung und Verlagerung von Hardwaretreibern aufgezeigt werden. Dabei werden die Erkenntnisse des paravirtualisierten Ansatzes mit einem nativen System verglichen. Alle Messungen wurden unter Verwendung des Infineon TriCore AURIX TC27X erstellt [47]. Jeder Rechenkern wird mit 200 MHz betrieben. Der Fokus der Messung liegt auf der Verwendung von Kommunikationscontrollern wie HighSpeed-CAN (500 kbit/s) und Fast Ethernet (100 Mbit/s). In allen Messungen werden Rechenkerne exklusiv für den Informationsaustausch genutzt und verarbeiten keine weiteren Lasten. Blockierende Aufrufe oder Sperrmechanismen werden vermieden. Die Stimulation erfolgt durch Anbindung externer Messgeräte, die Nachrichten über einen gemeinsamen Feldbus übertragen.

Abbildungen 7.5, 7.6 und 7.8 sind folgendermaßen aufgebaut: Native Systeme sind nicht paravirtualisiert und fungieren als Benchmark zum Vergleich aller Messungen.

```
1  void routeMessage(AppMultican_Message *canMsg){
2          ivmc_message_envelope me;
3          ivmc_message msg;
4
5  /* message targets for fieldbus transmission */
6          if(canRTable[canMsg->id][0]){
7                  AppCan_process_node0_msg(canMsg);
8          }
9
10 /* message targets on core1*/
11         /* ... */
12
13 /* message targets on core2*/
14         if(canRTable[canMsg->id][2]){
15                 msg.length = 12;
16                 msg.sourceCore = CORE2;
17                 msg.data = canMsg;
18
19                 me.payload = &msg;
20                 me.targetCoreMap = CORE2;
21                 me.request = IVMC_REQUEST_WRITE_PROD;
22
23                 HV_CallVirtualDevice(IvmcVdeId, &me);
24         }
25 }
```

Quelltext 7.2: Routing von CAN-Frames an Kern2 innerhalb des Backend-Treibers von DomM.

Zur Kommunikation zwischen VMs wird wiederum der IvmcVDE verwendet. Innerhalb der Messreihe *DomM→Feldbus* werden Nachrichten direkt von der privilegierten DomM über den verbundenen Feldbus versand. Diese Messung beinhaltet die physische Übertragungszeit über den Feldbus, inklusive aller Verwaltungskosten zum Routen einer Nachricht innerhalb der DomM. Für *DomU→Feldbus* werden Nachrichten von einer DomU bis hin zur DomM übertragen. Darauffolgend wird die eingehende Nachricht innerhalb der DomM über den verbundenen Feldbus versendet. Dies impliziert Verwaltungsaufwand zum Routen der Nachricht. Innerhalb des *DomU→DomU* Szenarios ist kein limitierender Feldbus involviert. Nachrichten werden innerhalb einer ECU von einer DomU über die DomM zu einer weiteren DomU geleitet. Hierbei müssen relevante Daten unter Verwendung des IvmcVDEs mehrfach in die jeweiligen Speicherbereiche der Ringpuffer einzelner VMs kopiert werden.

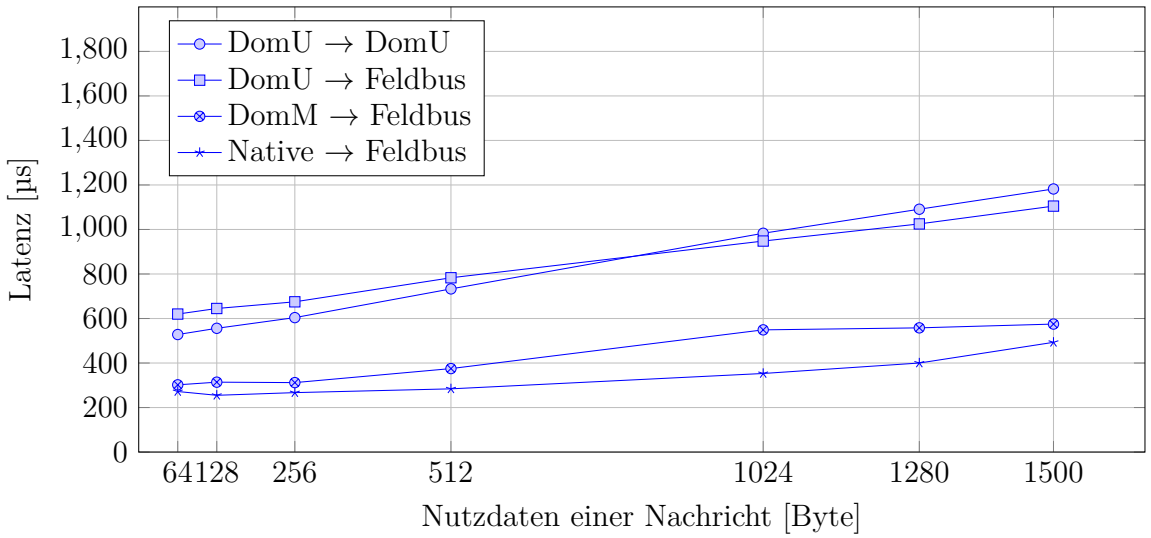


Abbildung 7.5: Domänenübergreifende Übertragung von Ethernet Frames zwischen virtuellen Maschinen.

7.5.1 Antwortzeit in Relation zur Nachrichtengröße

Im Folgenden werden die Antwortzeiten einzelner Nachrichten analysiert. Abbildungen 7.5 und 7.6 zeigen die Laufzeitmessungen, um eine Nachricht über einen CAN oder Ethernet basierenden Feldbus zu versenden. Es wird die Latenz in Abhängigkeit der Nutzdaten dargestellt. Innerhalb des Messaufbaus werden interrupt-basierte Kommunikationsmechanismen verwendet. Nachrichten werden grundsätzlich über die festgelegte DomM geroutet. Benötigte Kommunikationscontroller werden direkt der DomM zugewiesen (DDA). Diese besitzten somit exklusiven Zugriff auf die jeweiligen Hardwareressourcen und deren Register. Somit wird sowohl der Aufwand zur Paravirtualisierung der Treiberschicht innerhalb der DomM als auch zeitlicher Mehraufwand vermieden. Für die event-basierte Benachrichtigung eingehender Signale entsteht ein fixer zeitlicher Mehraufwand von 4 µs, um Interrupts vom HV an eine VM zu leiten [122]. Daher sind in allen folgenden Messungen, unter Verwendung von DDA, die zeitlichen Unterschiede zur Interruptverarbeitung zwischen nativen und paravirtualisierten Systemen geringfügig.

Abbildung 7.5 zeigt die Übertragung von Ethernet Frames, unter Verwendung des ICMP Protokolls (*ping*), um Echo-Anfragen mit unterschiedlicher Paketgröße zu vermessen. Im Vergleich zu CAN hat der Nachrichtenaustausch mit größeren Datenframes stärkeren Einfluss auf die Latenz. Dies ist darin begründet, dass Kopieroperationen von größeren Datenstrukturen rechenintensiver verarbeitet werden und somit größeren zeitlichen Aufwand, verursachen. Die Übertragung von Nachrichten zwischen VMs benötigt mindestens zwei Kopieroperationen. Somit erhöht sich gleichzeitig der zeitliche Aufwand sobald sich die Nutzdaten von Nachrichten erhöhen. Werden Frames zwischen DomUs und der DomM übertragen, wird Verwaltungsaufwand für den Nachrichtenaustausch innerhalb der IvmcVDE selbst erzeugt. Weiterhin fällt Rechen-

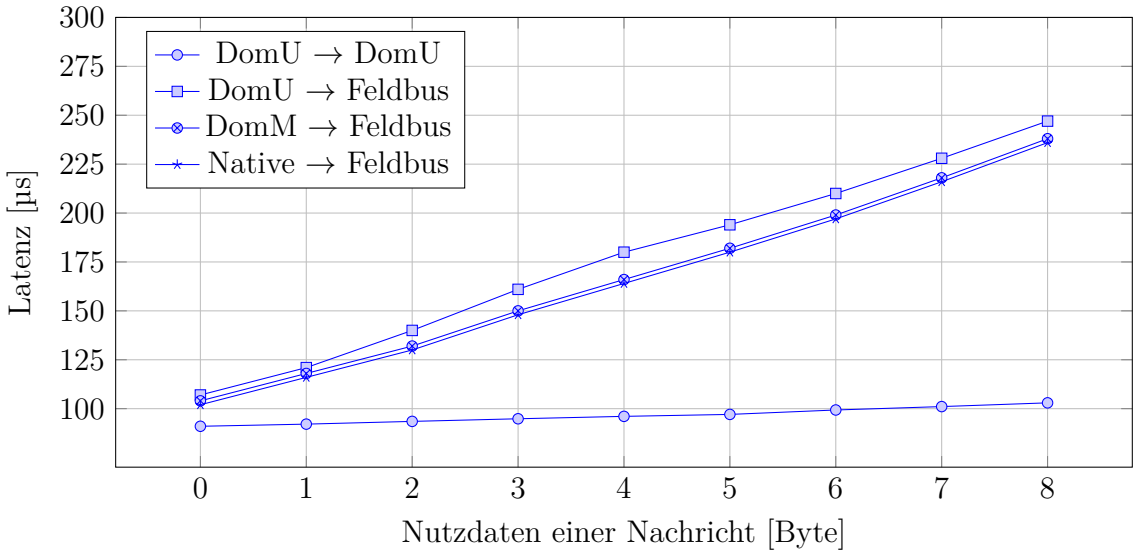


Abbildung 7.6: Domänenübergreifende Übertragung von CAN-Frames zwischen virtuellen Maschinen.

aufwand an, um zwischen den OSI-Schichten des TCP-Stacks zu abstrahieren und unter Verwendung einer SAT Nachrichten innerhalb jeder VM zu routen (vgl. die Zeitspanne zwischen *DomM→Feldbus* und *Native→Feldbus*).

Die Übertragung eines Ethernet Frames in einer Größe von 64 Byte, ausgehend von einer beliebigen DomU hin zum Feldbus, benötigt 620 µs. Die Übertragung von 1500 Byte Nutzdaten benötigt hingegen weniger als die doppelte Übertragungszeit. Für eine Nutzdatenrate ab ca. 700 Byte findet der Umbruch (Break-Even-Point) zwischen Latenzzeit statt, um Ethernet Frames zwischen DomUs oder einem angeschlossenen Feldbus auszutauschen.

Die Analyse der Antwortzeiten für CAN-Frames wird in Abbildung 7.6 illustriert und zeigt einen ähnlichen direkt-proportionalen Verlauf aller Messwerte. Begründet durch die vergleichbar kleinen Paketgrößen eines CAN-Frames haben Kopieroperationen zwischen VMs bzw. DomUs ohne Verwendung eines physischen Feldbusses keine größeren Auswirkungen auf die Nachrichtenlatenz. Dies verursacht einen leicht wachsenden zeitlichen Trend. Der Verwaltungsaufwand, um Frames innerhalb der DomM zu routen, verursacht einen zeitlichen Aufwand von ca. 1 µs. Hierbei wird eine statische Lookup-Tabelle (Routingtabelle bei CAN-Kommunikation) mit einer zeitlichen Komplexität von $\mathcal{O}(1)$ verwendet.

Die Verwendung des IvmcVDEs, um Nachrichten generisch zwischen DomUs zu übertragen, verursacht einen zusätzlichen zeitlichen Mehraufwand von ca. 4% (siehe Abbildung 7.6). Die längste Zeitspanne wird allerdings vom CAN-Controller selbst benötigt, um Nachrichten über den Feldbus zu übertragen. Die Dauer, um ein Frame zu erzeugen bis es dem CAN-Controller zum Versand übergeben wird, benötigt weniger als 5 µs. Die verbleibende Zeit wird innerhalb des CAN-Controllers für die physische Nachrichtenübermittlung über den Feldbus verwendet. Aufgrund kleiner Nachrichtengrößen eines CAN-Frames sind die Verwaltungskosten zur Übermittlung zwischen

VMs und DomUs erheblich höher, verglichen mit der Übertragung über einen angeschlossenen Feldbus. Es sei angemerkt, dass der Routingvorgang von Nachrichten innerhalb eines optimalen, paravirtualisierten Systems nahezu keine Auswirkung auf Antwortzeiten aufweist und somit fast lineare Messergebnisse liefert.

7.5.2 Durchsatzmessungen in Relation zur Nachrichtengröße

Zuerst wird die Bandbreite analysiert, um Informationen zwischen VMs auszutauschen. Hierfür wird die Datenrate zwischen einer DomU und der DomM gemessen. Dieses Vorgehen ist in Abbildung 7.7 dargestellt. Innerhalb dieses Szenarios entsteht noch kein Mehraufwand zum Routen von Nachrichten innerhalb der DomM, da lediglich der Kommunikationskanal und dessen Übertragungsrate vermessen wird. Dabei werden durch Polling Mechanismen geringfügig höhere Datenraten erzielt als im event-basierten Ansatz durch Interrupts. Werden Interrupts als Benachrichtigungssystem verwendet, entsteht zusätzlicher Rechenaufwand. Dies ist dadurch begründet, dass Interrupts durch den HV an jeweilige Rechenkerne bzw. VMs zugeteilt werden müssen. Werden kleinere Datenpakete versendet, ist die Datenrate grundsätzlich niedriger, verglichen zum Transfer von großen Datenpaketen. Dies liegt am statischen Rechenaufwand, um einzelne Anfragen durch den HV und zwischen VMs zu verarbeiten.

Kopieroperationen (wie *load* und *store*) von großen, aufeinanderfolgenden Datenstrukturen sind effizienter als die Verarbeitung von kleineren Datenframes (z.B. 7 Bytes Nutzdaten). Kleinere Datenpakete müssen aufgrund eines ungünstigen Byte-Alignments (8, 16, 32 Bit) unter Umständen Byte für Byte kopiert werden. Die verwendeten Ringpuffer besitzen eine Größe von 4096 Byte. Die Datenrate bricht bei 2049 Byte (inklusive 6 Byte PCI) ein. Ab dieser Nachrichtengröße kann der Ringpuffer nicht mehr als ein Frame auf einmal aufnehmen. Somit sind lediglich sequentielle Lese-und Schreibvorgänge auf den Ringpuffer möglich. Im dargestellten Messversuch wird gezeigt, dass die Implementierung des IvmcVDE es ermöglicht, Datenraten bis zu 97 Mbit/s bei einer Nachrichtengröße von 2047 Byte von einer VM zur anderen zu übertragen.

Nun soll der maximale Datendurchsatz der CAN und Ethernet Kommunikation ermittelt werden. Das gewählte event-basierte Verfahren verwendet Interrupts zur Benachrichtigung eingehender Informationen. In Abbildung 7.8a werden die Ethernet-Messergebnisse dargestellt. Native Systeme ohne Virtualisierungsschicht weisen im Vergleich zur Senderate eine doppelt so hohe Empfangsrate auf. Zum Empfang von Frames sind keine Kopieroperationen (*engl. zero-copy*) notwendig. Empfangene Frames werden per Direct Memory Access (DMA) in die Datenstrukturen einer VM kopiert. Um Frames zu versenden, muss jede Information in die Datenstruktur des Ethernet Controllers kopiert werden. Im Fall von *DomM→Feldbus* wird die Empfangsrate aufgrund eines zusätzlichen Kopiervorgangs innerhalb des Bridging-Mechanismus der DomM nahezu halbiert.

Im Rahmen der Messung von *DomU→DomU* werden die Daten innerhalb der DomM empfangen. Dort werden diese vom IvmcVDE in den lokalen Ringpuffer ko-

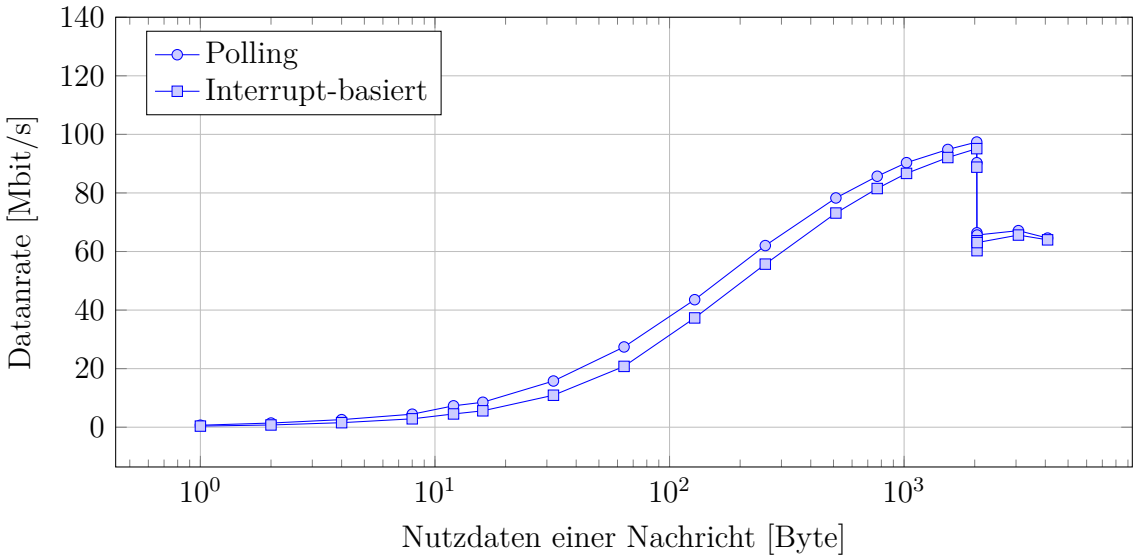
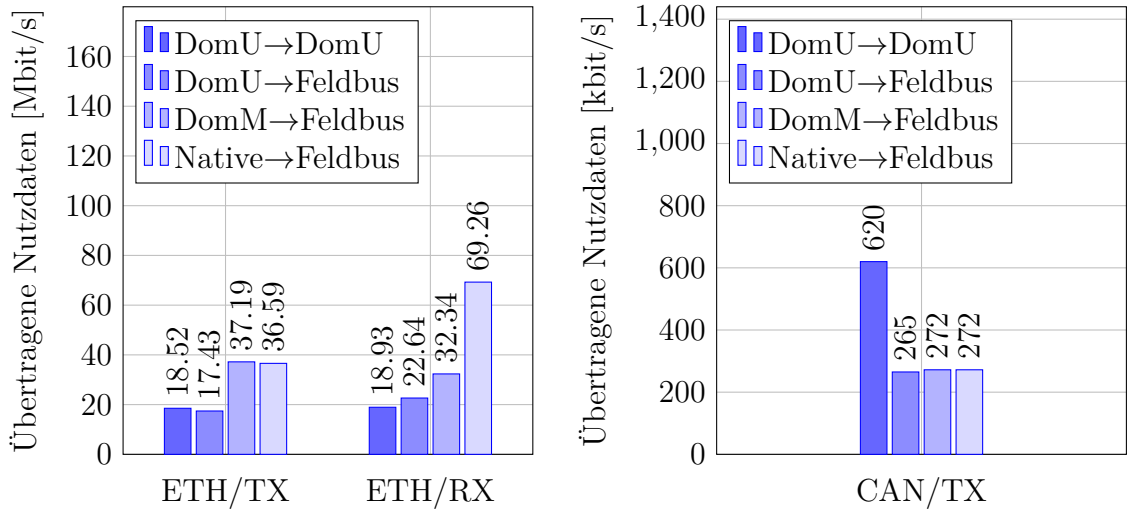


Abbildung 7.7: Maximale Datenrate zwischen virtuellen Maschinen [113, 126].



(a) Maximale Datenrate für Ethernet.

(b) Maximale Datenrate für CAN.

Abbildung 7.8: Gemessene Datenraten zwischen VMs unter Verwendung des IvmcVDE [68, 126].

piert. Im Folgenden wird ein Datenframe an eine weitere DomU geleitet. Hierfür wird wiederum der Mechanismus des IvmcVDEs verwendet, wodurch eine zweite Kopieroperation notwendig ist. Somit wird verständlich, dass hohe Datenraten (z.B. mit Ethernet) von der Anzahl der Kopieroperationen abhängig sind. Die Verwendung einer DomM hat signifikanten Einfluss auf die Performanz der Inter-VM-Kommunikation schwächerer Mikrocontroller. Verglichen mit einem nativen System erzielt die Verwendung des DomM-Ansatzes einen geringeren Datendurchsatz zwischen VMs. Dies ist anhand der Messreihe *DomU→Feldbus* und *DomU→DomU* in Abbildung 7.8a zu erkennen. Allerdings betragen die maximalen Nutzdaten für Ethernet Frames 1500 Bytes, wodurch meistens große Paketdaten versendet werden. Zusammengefasst existieren zeitliche Einschränkungen bei Verwendung von eingeschränkten Hardwareressourcen, sobald Ethernet Frames zwischen VMs oder einem verbundenen Feldbus durch eine DomM geroutet werden.

Für CAN Feldbusse ist die maximale Übertragungsrate durch die Übertragungsgeschwindigkeit des Hardwarecontrollers und durch die Bitrate des Feldbusses limitiert. Für die Datenraten, dargestellt in Abbildung 7.8b, werden hingegen selbst bei Maximalauslastung CAN-Frames schneller empfangen, als diese von vorhandenen Busteilnehmern abgesetzt werden können. Die Empfangsrate stellt somit keinen limitierenden Faktor dar. Dies bedeutet im Rückschluss, dass keine maximale Empfangsrate durch externe Messinstrumente feststellbar ist (keine Parallelkommunikation ist innerhalb des CAN-Busses zulässig). Während der Übertragung wurde kein Verlust (*engl. loss*) von Nachrichten detektiert. Somit existiert keine Messung zur maximalen Empfangsrate in Abbildung 7.8b.

Innerhalb eines nativen Systems ohne Virtualisierung kann eine maximale Sendegeschwindigkeit von 272 kbit/s bei einer statischen Nutzdatengröße von 8 Byte pro Frame erzielt werden. Ein identisches Ergebnis konnte für die Messreihe *DomM→Fieldbus* mit allen zusätzlichen Overheads erzielt werden. Dies begründet sich durch die langsamere Übertragungsrate des CAN-Controllers selbst, wodurch physikalisch keine schnellere Übertragungsgeschwindigkeit, unter Verwendung der angegebenen Bitrate des Feldbusses, möglich ist (siehe Kapitel 7.5.1). Alle Messergebnisse hierfür sind in Abbildung 7.8b veranschaulicht. Zusammen mit der PCI und den Nutzdaten besitzt ein CAN-Frame eine Größe von 108 Bit. Somit errechnet sich eine maximale Datenrate von 459 kbit/s.

Für die Messreihe *DomU→Fieldbus* werden wiederum ähnliche Ergebnisse erzielt. Die leichte Abweichung begründet sich sowohl durch den zusätzlichen Verwaltungsaufwand innerhalb der DomM als auch innerhalb des IvmcVDE zum Nachrichtenaustausch zwischen VMs. Um CAN-Frames zwischen DomUs austauschen zu können (*DomU→DomU*), wird wieder ein Bridging-Mechanismus innerhalb der DomM verwendet. Hierfür werden Antwortzeiten von 103 µs gemessen. Diese sind um ein Vielfaches schneller als eine Übertragung über den CAN-Feldbus. Somit können Datenraten mit einem maximalen Durchsatz von 620 kbit/s ermöglicht werden. Zusammengefasst hat, trotz Verwendung einer dezentralen DomM, ein softwaregesteuertes Routing von CAN-Frames innerhalb der Bridge des paravirtualisierten Systems, keinen großen Einfluss auf die Gesamtperformanz der Nachrichtenübertragung.

7.6 Zusammenfassung des Kapitels

Zur Verbesserung des Informationsaustausches zwischen VMs wurde innerhalb dieses Kapitels ein Kommunikationskonzept für einen unabhängigen Signalaustausch entwickelt. In diesem Rahmen wurde das vorgeschlagene Kommunikationskonzept implementiert und hierzu, unter Verwendung des Infineon TriCore AURIX Mikrocontrollers, Laufzeit- und Durchsatzmessungen vorgenommen. Signale werden zwischen VMs asynchron unter Kontrolle des HVs übermittelt. Dabei wird das Nachrichtenformat innerhalb der Nutzdaten der zu übertragenden Botschaften nicht festgelegt (siehe Kapitel 7.4.1). Somit ist das Kommunikationskonzept losgelöst von integrierten VMs und deren Softwaresystemen.

Im Vergleich zu nativen Systemen wird im dargestellten Lösungsansatz aufgrund zusätzlicher Kopieroperationen, die Übertragungsrate zwischen VMs reduziert. Dies wird bei hohen Datenraten, wie zum Austausch von Ethernet Datenpaketen, deutlich erkennbar. Um höhere Datenraten erzielen zu können, sind schnellere Hardwarecontroller notwendig. Hardwareseitige Virtualisierungsunterstützungen sind zur weiteren Steigerung dieser Datenraten von Vorteil. Werden hingegen CAN-Pakete übertragen, fallen dabei grundsätzlich kleinere Datenmengen an. Gemäß der dargestellten Messergebnisse sind die Datenraten eines typischen CAN-Feldbusses innerhalb paravirtualisierter Systeme weiterhin erzielbar. Im Vergleich zu anderen Feldbussen ist die erreichte Bandbreite zwischen VMs signifikant höher, da ECU-intern kein limitierender Feldbus vorliegt.

Weiterhin wurde die Partitionierung von Hardwaretreiber einzelner Kommunikationscontroller untersucht. Hierbei wurde die Verlagerung der Treiberschicht der AUTOSAR BSW innerhalb von VMs betrachtet. Unter Auflistung potentieller Integrationsszenarien relevanter MCAL Treibermodulen wurden Lösungskonzepte erarbeitet, in welchen gemeinsam genutzte Hardwaretreiber in unterschiedlichen Softwareschichten integriert sind. Für jedes Konzept wurde eigens eine individuelle Implementierung erstellt und beurteilt.

Speziell für sicherheitsrelevante Anwendungen wird vorgeschlagen, Hardwaretreiber innerhalb einer gesonderten VM zu isolieren. Hierfür wird eine privilegierte VM (genannt DomM) ausgewählt, um einzelne Treibermodule der betreffenden Kommunikationscontroller (z.B. CAN oder Ethernet) zu kapseln. Das Konzept, den MCAL der AUTOSAR BSW innerhalb einer virtualisierten Umgebungen in eine privilegierte VM zu verlagern, bietet somit die Möglichkeit, VMs unabhängig von deren Hardwaretreibern auszuführen. Dabei kann der Zugriff auf Treibermodule in einem abgeschlossenen Bereich sicherheitsrelevante bzw. nicht-sicherheitsrelevante Fahrzeugfunktionen voneinander trennen.

Die Kapselung der AUTOSAR MCAL-Schicht innerhalb einer speziellen VM (DomM) stellt eine aus Kostensicht vertretbare Lösung dar, um die Hardwaretreiber verbleibender VMs zu trennen. Für sicherheitsrelevante Systeme muss allerdings der potentielle Eintritt eines *Single Point of Failures* durch die DomM während der Sicherheitsqualifizierung berücksichtigt werden. Ist die DomM fehlerhaft, ist das gesamte System gefährdet, da benötigte Hardwaretreiber nicht weiter zur Verfügung

stehen. Um den jeweiligen ASIL erreichen zu können, sind somit sowohl erweiterte Qualifizierungsmaßnahmen laut ISO 26262 [89] als auch Absicherungsmaßnahmen wie Redundanzen notwendig. Dieses Vorgehen ist jedoch identisch zu einem klassischen sicherheitsrelevanten System, in welchem kein HV eingesetzt wird.

Um sowohl erwünschte Sicherheitsansprüche als auch notwendige Anforderungen an Echtzeitfähigkeit und Systemperformanz zu erfüllen, müssen daher gewisse Einschränkungen des Systems in Kauf genommen werden. Die erläuterte Methode - Verlagerung von Treibermodulen in eine gesonderte VM und Bildung virtualisierter Kommunikationskanäle innerhalb paravirtualisierter Systeme - ist hierfür ein valides Lösungskonzept. Allerdings erfordert dies zusätzliche Kommunikationspfade und weitere zeitliche Kopieraufwände aufgrund des asynchronen Datenaustauschs.

Die vorliegenden Konzepte sind nicht ausschließlich auf den verwendeten Mikrocontroller beschränkt und sind auf weitere Plattformen übertragbar. Ähnliche Konzepte zur Kapselung von Hardwaretreibern innerhalb einer privilegierten VM werden bereits innerhalb des Xen Hypervisors erfolgreich eingesetzt. Dieser ist unter anderem auch auf ARM basierten Mikrocontrollern integrierbar [33, 125].

8 Demonstratoraufbau zur Konsolidierung von virtuellen AUTOSAR-Softwaresystemen

Im Rahmen einer Machbarkeitsstudie soll nun der Aufbau eines Hochintegrationsdemonstrators zur Isolation von eingebetteten Fahrzeugsoftwaresystemen realisiert werden. Ziel des Demonstratoraufbaus ist eine zusammenfassende Evaluierung der zuvor entwickelten Konzepte und Methoden. Um die Umsetzbarkeit einer Virtualisierungslösung auf aktuellen Steuergeräten abschätzen zu können, sollen die zusätzlich anfallenden Laufzeitkosten eines eingebetteten HVs aufgezeigt werden. Hierfür wird bestehende Steuergerätesoftware mit unterschiedlicher Sicherheitseinstufung (ASIL) auf einer gemeinsamen Hardwareplattform hochintegriert [7]. Die bereits erzielten Ergebnisse und Lösungsansätze aus den vorhergehenden Kapiteln 4 und 5 zur Verwendung eines HVs werden hierfür vorausgesetzt. Weiterhin soll sowohl der in Kapitel 6 erläuterte nicht-intrusive Virtualisierungsansatz als auch die Kommunikationsmechanismen aus Kapitel 7 angewandt und auf die Bedarfe AUTOSAR-basierter Softwaresysteme portiert werden. Der verwendete HV der Firma ETAS Ltd. stellt wiederum die Isolation zwischen den jeweiligen Partitionen sicher. Um realitätsnahe Analyseergebnisse zu erhalten, werden zur Softwareintegration Fahrzeugfunktionen aus der Automobilindustrie verwendet. Der Infineon TriCore AURIX Mikrocontroller dient wiederum als Hardwareplattform. Als exemplarische Fahrzeugfunktionen werden die Systemfunktionen des BMW AUTOSAR Cores, eine sicherheitsrelevante Fahrzeugfunktion aus dem Bereich Powertrain und weitere proprietäre Hardwaretreiber integriert.

8.1 Herausforderung und Problemstellung

Wie in Kapitel 4 aufgeführt sind **Fahrzeugsoftwarekomponenten stark miteinander vernetzt** und können nicht ohne weitgehende Änderungsmaßnahmen aus bestehenden Softwarearchitekturen herausgelöst und verteilt werden. Dies betrifft auch die verwendete Fahrzeugsoftware für den Demonstratoraufbau.

Weiterhin benötigt die Zusammenführung von Funktionen **unterschiedlicher Sicherheitseinstufungen** sowohl zusätzliche Isolationsmechanismen der Software als auch der Hardware, um Rückwirkungsfreiheit garantieren zu können. Die bereits vorhandenen Mechanismen bestehender Steuergeräte, wie der Speicherschutz durch Verwendung einer MPU, werden hierbei übernommen oder durch den HV emuliert.

Darüberhinaus ist Steuergerätesoftware an feste Hardwareadressen gebunden und Programmcode wird an vordefinierte Speicherstellen integriert. Innerhalb aktueller Hardwareplattformen ist hierzu **keine MMU vorhanden**, um eine dynamische Speicheraufteilung vornehmen zu können. Diese Aufteilung muss daher vom HV realisiert werden.

Zur Migration von Fahrzeugsystemen werden Paravirtualisierungstechniken angewandt. Daher muss eine **Anpassung der Treibermodule** innerhalb des AUTOSAR MCALs durchgeführt werden. Hierbei wird die Ausführung unerlaubter Registerzugriffe durch Hypercalls ausschließlich im Kontext des HVs durchgeführt.

In E/E-Fahrzeugsystemen ist die **Echtzeitfähigkeit des Systems** von hoher Wichtigkeit. Diese darf nach Migration in eine virtualisierte Umgebung nicht beeinträchtigt werden. Das vordefinierte Zeitverhalten muss somit auch nach Portierung in eine virtualisierte Umgebung eingehalten werden.

Steuergeräte sind über diverse Fahrzeugbusse miteinander vernetzt. Durch **Emulation bestehender Bussysteme** muss das Kommunikationsverhalten paravirtualisierter Steuergeräte ECU-intern durch den HV nachgebildet werden. Emulierte Peripheriecontroller müssen somit im Vergleich zu echten Controllern ein identisches Verhalten aufweisen.

Da diese Arbeit den nicht-intrusiven Softwareintegrationsansatz verfolgt, soll die **Konfiguration der AUTOSAR BSW** einzelner virtualisierte Steuergeräte nicht verändert und vollständig übernommen werden. Ausschließlich notwendige Treiberschichten sollen im Rahmen notwendiger Paravirtualisierungsaufwände modifiziert werden.

8.2 Methoden zum Informationsaustausch zwischen virtualisierten AUTOSAR Partitionen

Diverse Integrationsansätze der AUTOSAR Treiberschicht und der Zugriffskontrolle einzelner Hardwareressourcen wurden bereits in Kapitel 7.3 diskutiert. Diese Methoden sollen nun zum Einsatz kommen und die Kommunikationsschnittstellen der einzelnen VMs an dem jeweiligen IvmcVDE angeschlossen werden. Dabei wurde der nicht-intrusive Ansatz verfolgt, bei welchem virtualisierte Systeme nach Möglichkeit nicht verändert werden sollen. Ziel ist es nun, Anpassungen an der BSW vorzunehmen, um den Nachrichtentransfer zwischen AUTOSAR-basierten VMs und der DomM weiter zu optimieren. Identifizierte Ansätze werden in Abbildung 8.1 dargestellt. Softwarekomponenten oberhalb der AUTOSAR RTE sollen dabei nicht verändert werden. Somit soll verhindert werden, dass während der Entwicklung von SWCs, Integrationswissen über partitionierte Softwareeinheiten und Kommunikationskanäle vorhanden sein muss.

Wie in Abbildung 7.4 dargestellt, werden Hardwaretreiber der AUTOSAR BSW in einer DomM gekapselt. Der Treiber des CAN-Moduls ist somit vollständig auf Kern 0 integriert. Dabei existieren unterschiedliche Anbindungsmöglichkeiten der AUTOSAR BSW an die Kommunikationsschnittstellen der IvmcVDE durch Frontend- bzw.

Backend-Treiber. Abbildung 8.1 stellt diese Integrationsansätze und deren Abstraktionsebenen nummeriert dar. Es wurden folgende Herangehensweisen identifiziert:

1. **Kommunikationsschnittstellen im MCAL**: Wie bereits in Abbildung 7.2 dargestellt, werden Frontend- bzw. Backend-Treiber innerhalb der AUTOSAR Treiberschicht integriert. Dabei sind die Funktionsaufrufe für den Empfang- und Sendevorgang einer Nachricht mit Hypercalls versehen und Nachrichten werden über den HV geleitet. Die grundsätzliche Konzeption des MCALs ist es, die Treiberschicht einer AUTOSAR BSW auf einfachem Wege, unter Verwendung vordefinierter Schnittstellen, austauschen zu können [15]. Somit soll eine BSW flexibel an unterschiedliche Hardwarecontroller angepasst werden können. Dies wird durch festgelegte Schnittstellen innerhalb der BSW ermöglicht. Bereits serialisierte Signale, welche für einen Feldbus bestimmt sind, werden an den HV und den jeweiligen IvmcVDE weitergeleitet. Diese Nachrichtenpakete werden von DomM empfangen und dort an einen externen Feldbus oder an andere DomUs geroutet.

   Vorteil dieser Methode ist, dass die Grundkonfiguration der AUTOSAR BSW nicht weiter angepasst werden muss. Dadurch ist der Paravirtualisierungsaufwand gering. Der bestehende MCAL kann leicht durch eine paravirtualisierte Variante ausgetauscht werden (siehe Kapitel 6.6.8). Alle Signale haben somit ein festgelegtes Datenformat bzgl. des betreffenden Feldbusses.

   Ein Nachteil dieser Herangehensweise ist die Übermittlung meist kleiner Nachrichtenpakete. Wie bereits in Abbildung 7.7 dargestellt, ist die Übertragung von kleinen Datenpaketen ineffizient. Daher sollten ECU-intern möglichst große Datenpakete transferiert werden, um den Kommunikationskanal bestmöglich ausnutzen zu können. Dies ist bei einer Übertragung von einzelnen CAN-Frames nicht möglich (siehe Abbildung 7.6). Weiterhin entsteht zusätzlicher Serialisierungs- bzw. Deserialisierungsaufwand, da Datenpakete den gesamten Kommunikationsstack der BSW für den Sende- und für den Empfangskanal durchlaufen müssen. Für ECU-externe Kommunikation über einen Feldbus ist dies notwendig, nicht aber für den ECU-internen Nachrichtenaustausch. Dort existiert bisher noch kein festgelegtes Nachrichtenformat.

2. **Kommunikationsschnittstellen im ECU Abstraction Layer**: Zur Verarbeitung von Nachrichten innerhalb der BSW wird der jeweilige Frontend-Treiber als zusätzliches HV-Interface (innerhalb des *Communication Hardware Abstraction*-Blocks) integriert [113]. Der PDU-Router ist innerhalb der BSW für die Zuordnung von I-PDUs (Signale) an die jeweilige Abstraktionsschicht eines Feldbussystems (wie CAN, FlexRay oder Ethernet) verantwortlich [15]. Unter Verwendung des PDU-Routers werden für den HV bzw. für die DomM bestimmte Signale als I-PDUs an das HV-Interface geroutet. Dieses Interface beinhaltet wiederum den Frontend-Treiber, welcher eingehende und ausgehende Signale weiterleitet.

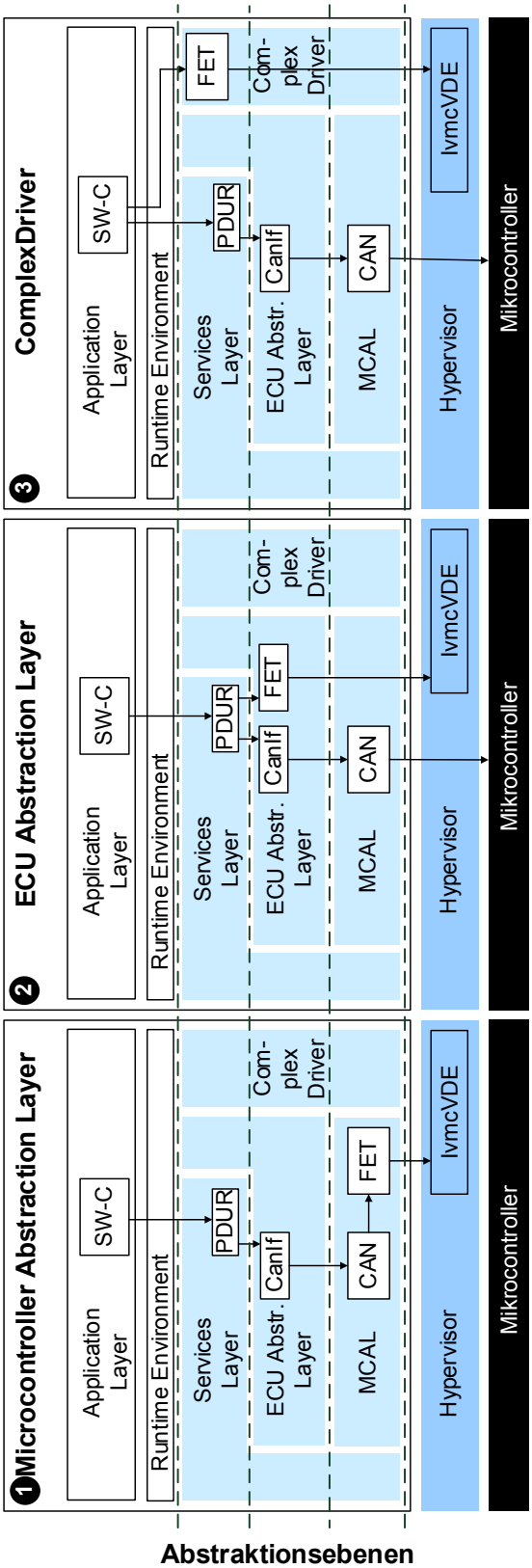


Abbildung 8.1: Ansätze zur Integration eines Schnittstellentreibers zum Nachrichtenaustausch mit einem Kommunikationskanal innerhalb des Hypervisors.

Der Vorteil dieser Herangehensweise ist, dass die Konfiguration der AUTOSAR RTE nicht angepasst werden muss. Eingehende Signale werden innerhalb des PDU-Routers unterhalb der RTE in der BSW verarbeitet. Dieses Verfahren eignet sich, falls Signale bereits innerhalb der VM selbst geroutet werden. Dies ist der Fall, sobald Peripheriecontroller nicht geteilt werden müssen und direkt mit der DomU verbunden sind.

Der Nachteil dieser Methode ist der massive Anpassungsaufwand innerhalb der BSW. Innerhalb des aktuellen AUTOSAR Standards in Version 4 ist die Virtualisierung von Systemen nicht vorgesehen. Daher ist der nachträgliche Eingriff in die BSW und das Hinzufügen eines Schnittstellenmoduls zum HV lediglich durch zusätzlichen Implementierungs- und Konfigurationsaufwand möglich. Weiterhin muss das Modul aufwendig in die BSW integriert werden, da eine Vielzahl von Schnittstellen und Abläufen zur Signalverarbeitung durch den AUTOSAR Standard fest vorgegeben sind. Auf Seiten der DomM ist dabei die Integration des jeweiligen AUTOSAR Kommunikationsstacks notwendig, um eingegangene Nachrichten entweder über einen Feldbus zu senden oder an andere DomUs weiterzuleiten.

3. **Kommunikationsschnittstellen als ComplexDriver**: Innerhalb jeder VM wird hierfür der Frontend-Treiber im Rahmen eines AUTOSAR ComplexDrivers integriert. Dieser wird auf selbiger Abstraktionsebene des Service Layers der BSW integriert. Der Treiber fungiert losgelöst von der AUTOSAR BSW als Schnittstelle für den Nachrichtenaustausch zwischen SWCs oberhalb der RTE und dem HV unterhalb des MCALs. Innerhalb des Frontend-Treibers werden wiederum Hypercalls verwendet, um Nachrichtenpakete an die jeweilige VDE-Instanz innerhalb des HVs zu übertragen [113].

   Der Vorteil dieser Herangehensweise ist, dass der Quelltext der AUTOSAR BSW und der integrierten MCAL-Schicht nicht modifiziert werden muss. Weiterhin erfolgt die Signalverwertung ohnehin auf höherer Abstraktionsebene. Dadurch werden Informationsdaten nicht weiter für einen entsprechenden Feldbus aufgeteilt und größere Datenpakete werden gebildet. Ist bereits bekannt, dass bestimmte Signale ausschließlich für den ECU-internen Signalaustausch bestimmt sind, kann der Serialisierungsaufwand durch die AUTOSAR BSW eingespart werden.

   Einer der Nachteile dieser Methode ist der Implementierungsaufwand des ComplexDrivers. Weiterhin muss die AUTOSAR RTE neu konfiguriert werden, sodass notwendige Kommunikationsbeziehungen zwischen SWCs und dem jeweiligen Frontend-Treiber verbunden sind. Somit ist ein gewisses Vorwissen auf Seiten des Systemintegrators notwendig, um Signale korrekt zuordnen zu können. Wurde innerhalb der DomU noch keine Unterscheidung zwischen ECU-internen bzw. -externen Signalen getroffen, muss dies innerhalb der DomM vorgenommen werden. Zum Puffern eingehender Datenpakete entsteht zusätzlicher Rechenaufwand und Speicherbedarf.

Die drei erläuterten Szenarien zeigen auf, dass innerhalb der schichten-basierten AUTOSAR BSW unterschiedliche Ansätze zur Verbindung von Kommunikationsschnittstellen existieren. Zur besseren Beurteilung wurde für jedes Szenario eine eigene Implementierung erstellt und evaluiert. Dabei wird grundsätzlich beurteilt, in welcher Art und Weise in die Funktionsweise der BSW eingegriffen und deren Quellcode modifiziert werden darf.

Wird eine Kommunikationsschnittstelle **innerhalb des MCALs** integriert, muss die BSW am wenigsten modifiziert werden. Allerdings entsteht dabei der größte Serialisierungs- bzw. Deserialisierungsaufwand, um Nachrichten zwischen den Schichten der BSW zu übertragen (siehe Abbildung 7.2).

Werden Nachrichten **innerhalb des ECU Abstraction Layers** geroutet, können große Datenpakete gebildet und effizient übermittelt werden. Allerdings entsteht hierbei signifikanter Integrationsaufwand, um in die Funktionsweise der BSW einzugreifen [113].

Ist eine **Kommunikationsschnittstelle als ComplexDriver** integriert, erfolgt die Modulintegration vollständig außerhalb der BSW. Der Kommunikationskanal der verbleibenden DomUs mit einem Feldbus wird somit über DomM und über dessen Backend-Treiber geleitet, welcher auch als ComplexDriver realisiert ist. Trotzdem ist eine erneute Konfiguration der RTE notwendig, um I-PDUs der SWCs auf die neue Schnittstelle umzuleiten.

In Anbetracht des erhöhten Integrationsaufwands der zur Verfügung stehenden Fahrzeugsoftware soll die bestehende Konfiguration der BSW inklusive des MCALs wiederverwendet werden. Daher werden die Nachrichten zur Kommunikation zwischen VMs durch die **Integration eines ComplexDrivers** an den HV geleitet.

8.3 Aufbau eines Hochintegrationsdemonstrators

Im folgenden Demonstratorbeispiel sollen mehrere AUTOSAR-basierten Softwaresysteme auf die verwendete Hardwareplattform, den Infineon TriCore AURIX TC275, konsolidiert werden. Dieser ist auf einem Evaluationsboard (*TFT-APPLICATION-KIT*) angebracht, welches über einen TFT-Monitor verfügt. Unter Verwendung eines Touchscreens können Benutzereingaben an den Mikrocontroller weitergeleitet und gleichzeitig aktuelle Systemzustände dargestellt werden (siehe Abbildung B.6).

8.3.1 Softwarearchitektur und portierte Fahrzeugsoftware

Der Schichtenaufbau der integrierten Softwarearchitektur ist in Abbildung 8.2 dargestellt. Integrierte VMs enthalten dabei AUTOSAR-basierte Softwaresysteme. Jede VM bildet wiederum eine Partition und wird exklusiv einem Rechenkern zugeordnet (siehe Kapitel 6.6.2). Weiterhin dient der in Kapitel 6.6 analysierte HV der Firma ETAS Ltd. zur Abschottung der jeweiligen VMs. Dabei sind alle Softwarekomponenten oberhalb der AUTOSAR RTE integriert. Die AUTOSAR BSW (*CUBAS*) und das zugehörige OS (*RTA-OS*) werden außerdem von ETAS Ltd. bzw. der Robert Bosch GmbH zur Verfügung gestellt.

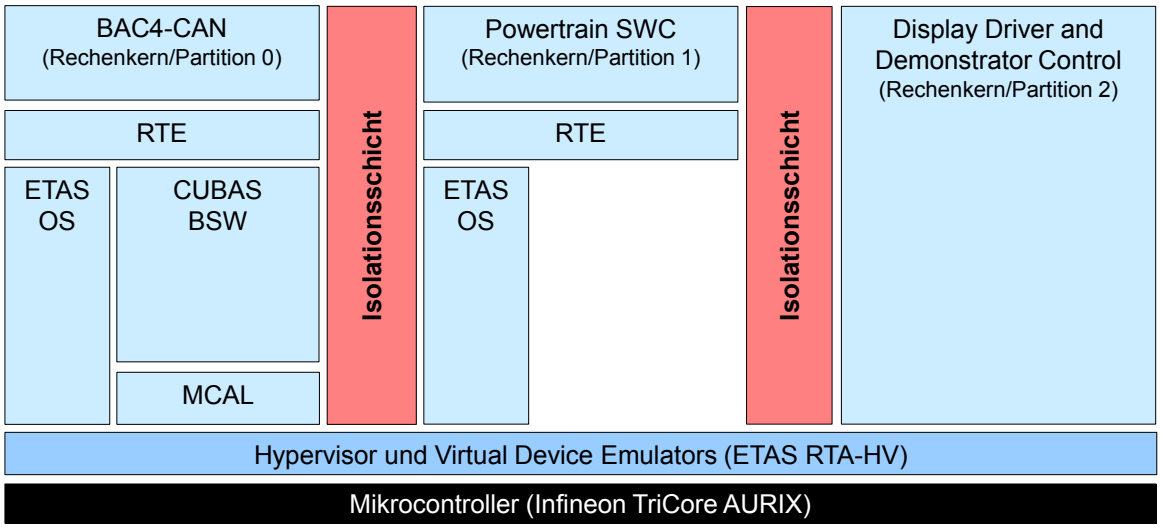


Abbildung 8.2: Softwarearchitektur des Hochintegrationsdemonstrators zur Konsolidierung von unterschiedlichen Fahrzeugsoftwarepartitionen.

Zusammen mit dem Mikrocontroller-zugehörigen MCAL, zur Verfügung gestellt von Infineon Technologies AG, sind alle BSW-Module und das OS unterhalb der RTE integriert. Benötigte Treibermodule werden hierfür paravirtualisiert. Dies betrifft auch die grafische Displayausgabe, die als proprietären Softwarepart innerhalb einer VM isoliert wird.

Partition 0 integriert den BMW AUTOSAR Core 4, der die Systemfunktionen eines Fahrzeugs beinhaltet. Beispiele für Systemfunktionen sind Softwarekomponenten zur Umsetzung von Diagnoseanfragen oder zur Kodierung bzw. Parametrisierung eines Steuergerätes. Somit befindet sich der MCAL vollständig innerhalb dieser Partition und stellt somit die in Kapitel 7.3 DomM dar. Der Integrationsaufwand wurde bereits in Kapitel 6.6.8 analysiert. Weiterhin ist der Kommunikationsaufbau bis zum benötigten CAN-Modul integriert.

Partition 1 beinhaltet eine Softwarefunktionalität der Domäne Powertrain zur Steuerung des Energiemanagements im Fahrzeug (*fEPM - flexible Energy and Power Management* [1]). Sie spiegelt eine sicherheitsrelevante Softwarekomponente wider. Um Rückwirkungsfreiheit garantieren zu können, soll diese Funktion innerhalb einer VM vom restlichen System abgeschottet werden [89]. Gleichzeitig kann durch integrierte Testumfänge die Funktionalität der Komponente veranschaulichen werden (z.B. Zustandsansteuerung zur energetischen Klemmensteuerung). Weiterhin ist paravirtualisierte OS mit dem HV verbunden. Somit verwenden beide VMs das identische OS, die mit unterschiedlichen Konfigurationen betrieben wird. Zur Initialisierung der Hardware wurde die Funktionalität des MCU Moduls deaktiviert. Lokale Hardwareressourcen eines Rechenkerns werden direkt angesprochen (DDA).

Partition 2 isoliert den Displaytreiber des Touchscreens vom verbleibenden System. Innerhalb dieser Partition werden alle Anwendungen integriert, die eine Benutzerinteraktion erlauben. Das Display selbst wird hierbei durch eine SPI-Schnittstelle mit dem Evaluationsboard verbunden und direkt per DDA angesteuert. Jegliche Kommu-

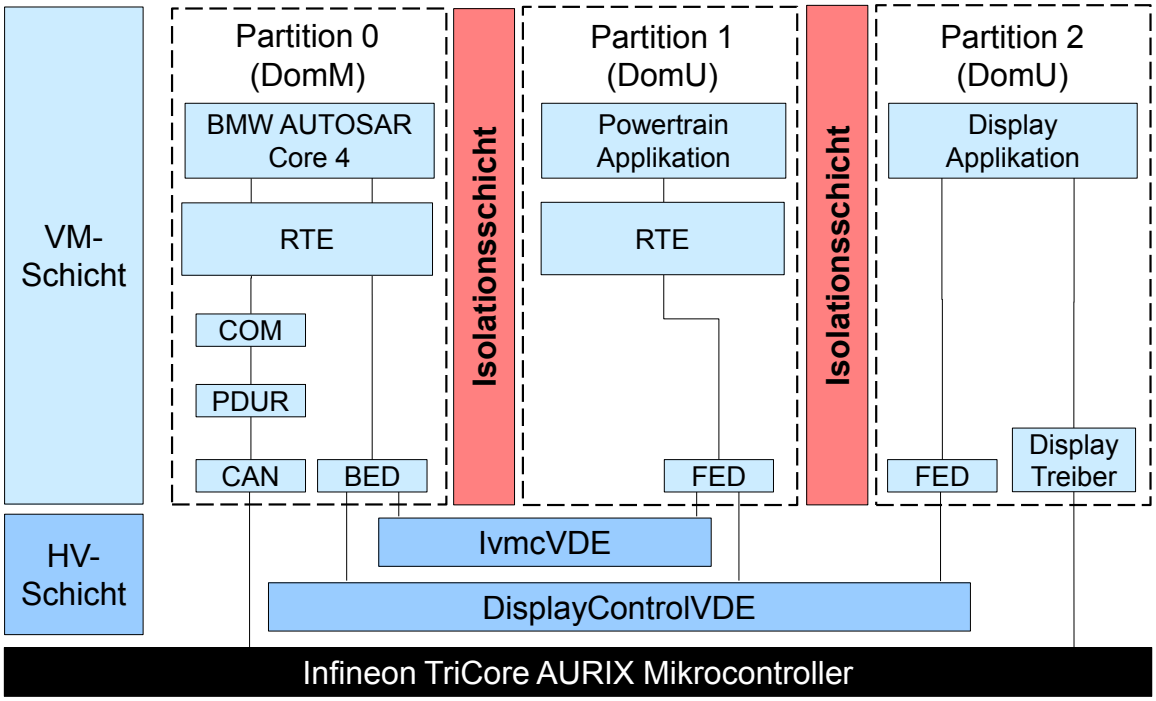


Abbildung 8.3: Kommunikationsmöglichkeiten zwischen virtuellen Maschinen innerhalb des Demonstratoraufbaus.

nikation zwischen der VM des Displaytreibers und dem verbleibenden System wird durch den HV kontrolliert. Sind empfangene Displayeingabewerte korrekt plausibilisiert, werden diese an die verbleibenden VMs weitergeleitet.

8.3.2 Informationsfluss zwischen virtuellen Maschinen

Die Kommunikation zwischen VMs ist in Abbildung 8.3 dargestellt. Zur Realisierung der Kommunikationskanäle werden mehrere VDEs verwendet. Wie in Kapitel 7.4 erläutert, werden diese durch Frontend- und Backend-Treiber innerhalb der VMs mit dem HV verbunden (siehe FET und BET ComplexDriver in Abbildung 8.3). Der erste Anwendungsfall ist die Emulation der Interaktion zwischen Partition 0 und 1 (durch IvmcVDE). Botschaften werden lediglich zwischen Partition 0 und 1 ausgetauscht. Dadurch ist das Kommunikationsprotokoll wesentlich vereinfacht.

Ein weiterer VDE namens *DisplayVDE* ist für den Austausch der Steuerungsbotschaften des TFT-Displays bestimmt. Dieser versorgt einerseits Softwarekomponenten innerhalb der Partitionen mit Informationen über potentielle Displayeingaben. Andererseits werden die Zustände einzelner Softwarekomponenten an Partition 2 und den jeweiligen Displaytreibern übermittelt, um Programmausgaben am TFT-Display darzustellen. Die Ansteuerung von *DisplayVDE* wird wiederum von dem VM-individuellen Frontend-Treiber umgesetzt.

Jegliche Interaktion mit externen Feldbussen wird über das CAN-Modul innerhalb von Partition 0 verarbeitet. Die Schnittstellen des Treibermoduls sind mittels DDA

direkt mit dem CAN-Peripheriecontroller verbunden. Eingehende bzw. ausgehende Signale werden somit zuerst an Partition 0 geleitet und innerhalb der AUTOSAR BSW verarbeitet. Sind CAN-Signale für andere Steuergerätepartitionen bestimmt, müssen diese unter Verwendung des Backend-Treibers und des zugehörigen VDEs an diese weitergeleitet werden.

8.3.3 Initialisierung der Hardware

Zur Konfigurationsphase während des Startvorgangs eines Steuergerätes können Konflikte beim Zugriff auf gemeinsame Hardwareressourcen entstehen (z.B. durch Mehrfachinitialisierungen). Somit sollten Peripheriemodule an einer zentralen Stelle konfiguriert werden. Ein typischer Anwendungsfall ist das Neustarten bzw. Herunterfahren einer dedizierten VM im System aufgrund eines Fehlers. In Kapitel 6.6.8 wurde bereits die Integration grundlegender AUTOSAR MCAL-Module erläutert und auf potentielle Problemstellungen hingewiesen. Greifen mehrere VMs auf gemeinsame Hardwareressourcen zu, können diese, wie in Kapitel 7.3 beschrieben, müssen diese durch einen VDE im HV emuliert oder innerhalb einer gesonderten VM (z.B. DomM) isoliert werden. In den meisten Fällen muss zumindest die Konfiguration dieser Module angepasst bzw. erweitert werden. Existiert kein gemeinsamer Zugriff auf Hardwareressourcen, können diese direkt mit der jeweiligen VM verbunden werden (DDA) und müssen nicht geteilt bzw. emuliert werden.

Der MCU-Treiber ist innerhalb der AUTOSAR BSW für die Initialisierung der Hardware (Peripheriecontroller, CPU-Konfigurationen, Resets, etc.) zuständig [25]. Er ist somit in jedem Steuergerät vorhanden und kann nicht ohne Konfigurationsaufwände geteilt werden. Der MCU-Treiber ist eng mit dem ECU State Manager-Modul verbunden (siehe Abbildung 8.5). Er ist der erste Treiber, der vom EcuM aufgerufen wird [10]. Dabei werden in erster Linie die Hardwareeinstellungen für die nachfolgende Treiberinitialisierungen vorgenommen.

Die in Abbildung 8.4 dargestellte Reihenfolge der Funktionsaufrufe innerhalb des EcuMs dient zur geordneten Initialisierung der Hardwarekomponenten vor Start des OS. Dabei ist *EcuMDriverInitListZero()* in diesem Fall nicht befüllt und beinhaltet lediglich Module, die bereits vor Hinzunahme der PostBuild-Konfiguration initialisierbar sind. Die Funktion *Mcu_Init()* initialisiert den MCU-Treiber, wodurch die Einstellungen für nachfolgende Treiber geladen und grundlegende Mikrocontrollereinstellungen vorgenommen werden (z.B. Konfiguration der Phase Lock Loops (PLLs)). Erst in *EcuMDriverInitListOne()* werden Hardwaretreiber (wie CAN, GPT, DIO, etc.) in festgelegter Reihenfolge mit der zuvor geladenen Konfiguration initialisiert. Die Funktion *EcuMDriverRestartList()* ist optional und dient zur Initialisierung von Treibern bei wiederkehrender Reaktivierung des Systems. Im Nachgang werden die Interrupt-Vektortabellen des OS geladen. Durch Aufruf der Funktion *StartOS()* wird das OS gestartet.

Innerhalb eines klassischen AUTOSAR Systems werden alle Initialisierungslisten innerhalb des EcuM-Moduls zur Konfiguration der Hardwarekomponenten durchlaufen. Besteht ein Konflikt aufgrund einer gemeinsam verwendeten Hardwareressource,

EcuM_Init(): Initialisierungsroutine des ECU State Manager.
- ↳ **EcuMDriverInitListZero()**: Treiberinitialisierung vor PostBuild und OS-Start.
- ↳ **Mcu_Init()**: Gesonderter Aufruf zur Initialierung des MCU-Treibers.
- ↳ **EcuMDriverInitListOne()**: Treiberinitialisierung vor OS-Start.
- ↳ **EcuMDriverRestartList()**: Reinitialisierung bei WakeUp (optional).
- ↳ **InitializeVectorTable()**: Initialisierung der Interrupt-Vektortabellen.
- ↳ **StartOS()**: Start des AUTOSAR-Betriebssystems.

Abbildung 8.4: Grundlegender Startup-Prozess zur Hardwareinitialisierung eines AUTOSAR-basierten Systems.

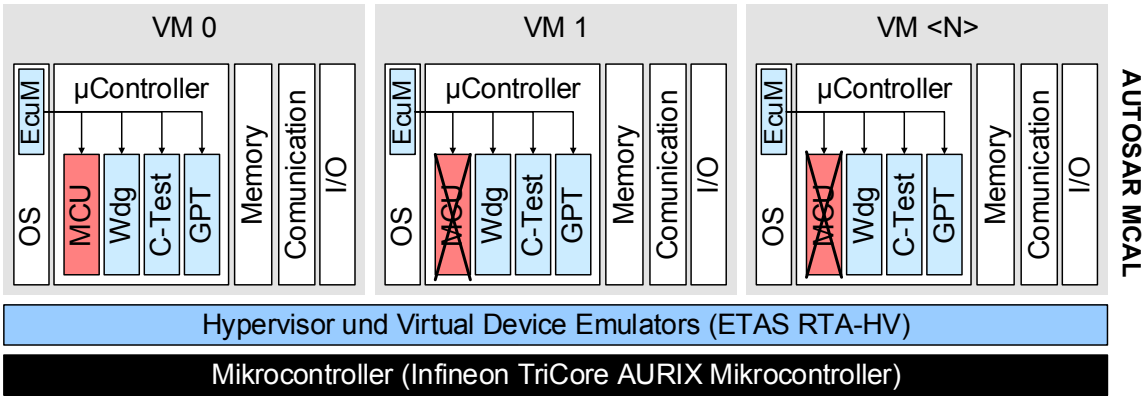


Abbildung 8.5: Initialisierung virtualisierte Steuergeräte unter Verwendung des MCU-Moduls.

muss dies während der Initialisierungsphase berücksichtigt werden. Zusätzlich betrifft dies die unterschiedlichen Konfigurationen der Kommunikationscontroller. Im Fall eines CAN-Controllers entscheidet die Festlegung der Bitrate über die jeweilige Busgeschwindigkeit. Diese kann einmalig im CAN-Controller justiert werden. Wurde das vorherige Steuergerät mit einer anderen Busgeschwindigkeit konfiguriert, muss dies zur tatsächlichen Konfiguration des CAN-Controllers berücksichtigt werden.

Problematischer ist die Konfiguration der CPU und anderer kernlokaler Komponenten durch den MCU-Treiber. Falls diese in einem vorhergehenden Steuergerät anderweitig vorbelegt waren, könnten diese zu einem unerwünscht zeitlichem Ablaufverhalten führen. Ein Beispiel hierfür sind unterschiedlich konfigurierte Frequenzen der Rechenkerne. Im Fall des AURIX Mikrocontrollers können diese nicht mit unterschiedlicher Taktrate betrieben werden. Zur Hochintegration mehrerer Steuergeräte ist hierfür die Prozessor- bzw. Systemkonfiguration zu homogenisieren.

Werden Steuergeräte auf virtualisierte Plattformen überführt, müssen der MCU-Treiber und die Initialisierungslisten innerhalb jeder VM angepasst bzw. entfernt werden (siehe Abbildung 8.5). Andernfalls werden zuvor angelegte Hardwareeinstellungen von nachfolgenden VMs während der Startphase überschrieben. Eine Aufteilung der Konfiguration zwischen VMs ist nicht möglich. In diesem Fall ist die Konfiguration der

zuletzt gestarteten VM gültig. Verbleibende VMs werden dadurch mit nicht korrekten Hardwareeinstellungen betrieben.

Alternativ kann die Konfiguration des MCU-Moduls an einer gesonderten Stelle innerhalb einer DomM stattfinden. Geteilte Hardwareressourcen werden dort direkt konfiguriert und in Betrieb genommen. Können Hardwareressourcen (wie Peripheriecontroller) direkt mit einer VM verbunden werden, liefert die DomM die notwendigen Konfigurationsdaten an die betroffene DomU. In beiden Fällen sind diese in den Initialisierungslisten innerhalb der EcuM-Moduls zu berücksichtigen. Dabei sind Initialisierungsschritte betroffener Hardwaretreiber zu ignorieren.

8.4 Untersuchung des Demonstratoraufbaus

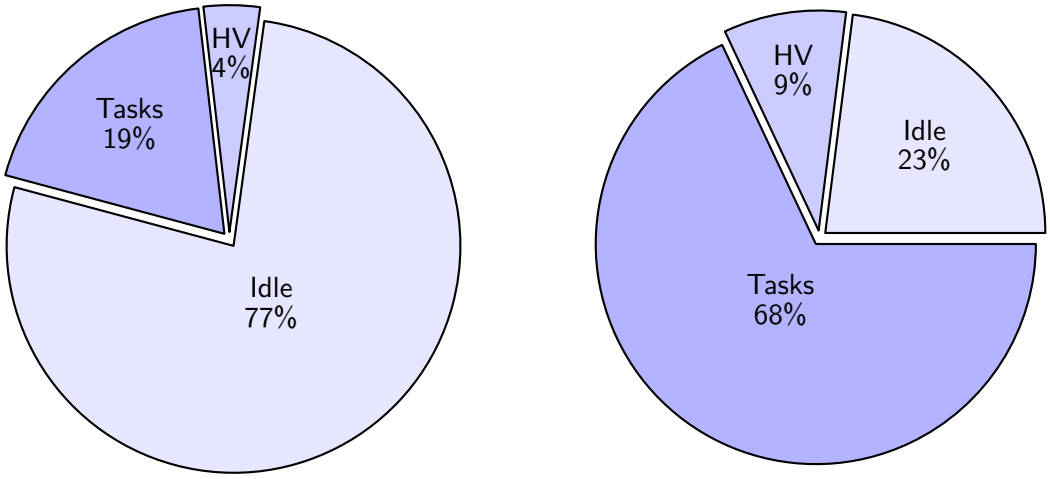
Nun soll eine quantitative Aussage des benötigten Rechenbedarfs zur Virtualisierung von AUTOSAR-basierten Steuergeräten im Rahmen des Demonstratoraufbaus getroffen werden. Der Portierungsaufwand einer minimalistischen Variante des AUTOSAR MCALs wurde bereits in Kapitel 6.6.8 beschrieben. Dabei wurde speziell auf die Vorgehensweise zur Paravirtualisierung der jeweiligen Module eingegangen. Die Integrationskosten bzw. der Portierungsaufwand wurde in Kapitel 6.6.8 abgeschätzt.

8.4.1 Analyse des Rechenbedarfs

Im Folgenden wird der prozentuale Anteil dargestellt, in welcher Softwareschicht (HV, VMs oder Idle-Modus) Rechenleistung im laufenden Systembetrieb verbraucht werden. Zur Ermittlung dieser Werte wird speziell Partition 0 und 1 betrachtet, da diese die relevanten Fahrzeugfunktionen beinhalten. Abbildung 8.6 zeigt die gemessene Rechenlast von Partition 0 und Partition 1. Diese wurde über einen Zeitraum von 60 Sekunden ermittelt.

Abbildung 8.6a zeigt den Rechenbedarf des BMW AUTOSAR Cores. Dabei wird ersichtlich, dass 19% der Kernauslastung von Systemfunktionen in Partition 0 innerhalb derer Taskverarbeitung beansprucht werden. Weitere 4% der Rechenkapazitäten werden durch die Verarbeitung im HV benötigt. In 77% der gemessenen Zeit verbleibt der Rechenkern im Idle-Modus. Dies ist damit begründet, dass Systemfunktionen typischerweise reaktive Systeme darstellen und eingehende Dienstanfragen anderer SWCs verarbeiten. Dieses Verhalten verursacht also wenig statischen Rechenaufwand.

Abbildung 8.6b zeigt die Rechenbedarfe der integrierten Fahrzeugfunktion zugehörig zur Powertrain Domäne. Dabei werden von dieser Funktion 68% der verfügbaren Rechenkapazitäten beansprucht. Weiterhin werden 9% der Kernlast innerhalb der HV-Schicht verbraucht. Die verbleibenden 23% der Rechenkapazitäten ist der Rechenkern im Idle-Modus. Der hohe Rechenbedarf der Fahrzeugfunktion wird aufgrund zyklischer, schnell aufeinanderfolgenden Taskaufrufe verursacht. Somit entsteht eine hohe, stetige Beanspruchung des Rechenkerns.



(a) Partition 0: Systemfunktionen des BMW AUTOSAR Cores 4.

(b) Partition 1: Fahrzeugfunktion der Domäne Powertrain.

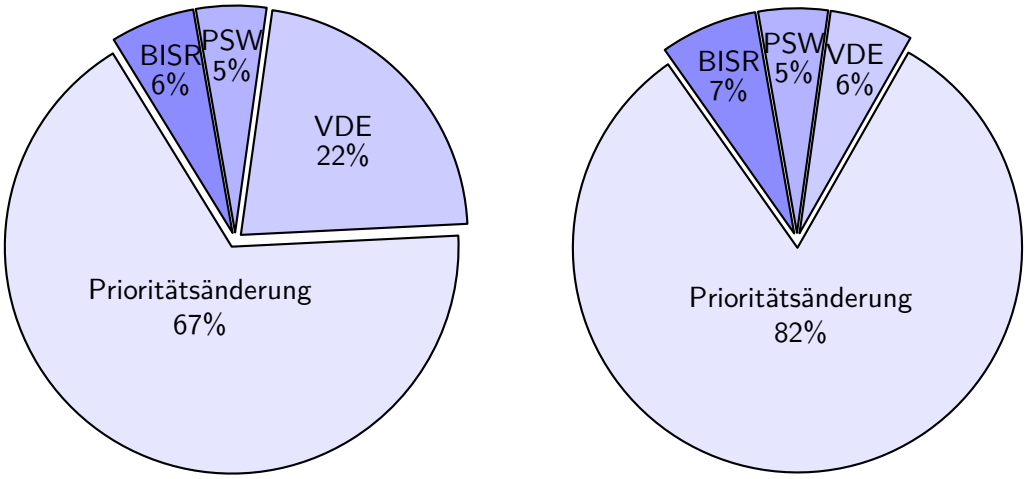
Abbildung 8.6: Auslastung der Rechenkerne in Partition 0 und 1 innerhalb des Demonstratoraufbaus gemessen über einen Zeitraum von 60 Sekunden [7, 64].

8.4.2 Funktionsbeanspruchung innerhalb des Hypervisors

Nun wird die Funktionsbeanspruchung des HVs innerhalb der HV-Schicht weiter aufgeschlüsselt (Registerzugriffe, Prioritätsänderungen und VDE-Aufrufe). Abbildung 8.7 zeigt die Auswertung auftretender Hypercalls und die damit verbunden Rechenbedarfe. Diese wurden über einen Zeitraum von 60 Sekunden gemessen.

Abbildung 8.7 zeigt die Registerzugriffe durch den HV. Zuerst werden Zugriffe auf das PSW ermittelt, die im Vergleich zu anderen Posten für beide Partitionen 5% aller Zugriffe betragen. Diese sind unter anderem notwendig, um die Zugriffsrechte (Kernel- oder User-Modus) der zugehörigen VM umzuschalten. Ähnliche Werte werden durch Zugriffe auf das BISR-Register verursacht. Diese Zugriffe verursachen 6% für Partition 0 und 7% für Partition 1 der Rechenbedarfe innerhalb der HV-Schicht. Das BISR-Register dient zur Aktivierung bzw. Deaktivierung von Interrupts, speichert den unteren Kontext des aktuellen Tasks und speichert unter anderem die Priorität (CCPN) der aktuellen CPU.

Den signifikantesten Anteil der HV-Aufrufe verursacht die Veränderung der CPU-Priorität durch den HV. Dies ist bei allen Taskwechseln zur Priorisierung eingehender Interrupts notwendig, um atomare Aufrufe garantieren zu können (siehe Kapitel 6.6.3). Diese Funktionalität wird innerhalb des paravirtualisierten OS umgesetzt. Daher beträgt der relative Anteil dieser Aufrufe 67% für Partition 0 und 82% für Partition 1. Ein nicht-unterbrechbares Verhalten wird auch während des Aufrufs von VDEs und derer Verarbeitung benötigt. Die Häufigkeit dieser Aufrufe beträgt 22% in Partition 0 und 6% in Partition 1. Ursache hierfür ist der erhöhte Routing-Aufwand durch den HV aufgrund einer Vielzahl kommunizierender Softwarekomponenten.



(a) Partition 0: Systemfunktionen des BMW AUTOSAR Cores 4.

(b) Partition 1: Fahrzeugfunktion der Domäne Powertrain.

Abbildung 8.7: Funktionsbeanspruchung durch Registerzugriffe des Hypervisors in Partition 0 und 1 innerhalb des Demonstratoraufbaus, gemessen über einen Zeitraum von 60 Sekunden [7, 64].

8.4.3 Beurteilung der Messergebnisse

Für den Demonstratoraufbau entstand hierbei zusätzlicher Konfigurationsbedarf, besonders innerhalb des HVs. Die Grundkonfiguration der AUTOSAR BSW inklusive des OS wurde dabei nicht verändert. Hierbei konnten Migrationsaufwände eingespart werden. Wie bereits in Kapitel 6.6.8 dargelegt, musste der Source Code der Treiberschicht und des OS paravirtualisiert werden, wodurch zusätzlicher Integrationsaufwand entstand. Zur Abbildung des ursprünglichen Kommunikationsverhaltens waren zusätzliche Implementierungsaufwände notwendig, um virtualisierte Fahrzeugbusse innerhalb der genannten VDEs individuell zu erstellen. Dabei ist die Struktur von VDEs und Schnittstellen zwischen VMs durch den IvmcVDE realisiert.

Wie bereits in Voruntersuchungen von Kapitel 6 erörtert, existieren zeitliche Einbußen für virtualisierte Systeme. Diese hängen von der eingesetzten Hardware ab und müssen für neue Controller individuell bewertet werden. Diverse Registerzugriffe, welche im Kernel-Modus erfolgen müssen, benötigen erhöhte Bearbeitungszeiten durch den HV. Es wurde in Abbildung 8.7 dargelegt, dass die Interruptlatenz (siehe Kapitel 6.6.5) für aktuell integrierte Systeme noch ausreichend. Allerdings ist die momentane Verarbeitung der Interrupts im System bei vielen Unterbrechungen nicht zuverlässig. Schnell aufeinanderfolgende Interrupts können innerhalb dieser Virtualisierungslösung nicht rechtzeitig bearbeitet werden und das Echtzeitverhalten beeinträchtigen [92].

Zusätzlich muss der Mehraufwand für Kontextwechsel berücksichtigt werden, welcher sich zusätzlich negativ auf die dargelegten Rechenbedarfe auswirkt (siehe Kapitel 6.6.5). Es muss also zur Hochintegration von E/E-Systemen geprüft werden, ob das Echtzeitverhalten nach einer Zusammenfassung noch haltbar ist [92].

Nicht zuletzt muss der benötigte Rechenaufwand einer zusätzlichen HV-Schicht zur Softwareintegration einkalkuliert werden [92]. Wie in Abbildung 8.6 dargestellt, verursacht der HV ca. ein Fünftel des auftretenden Rechenbedarfs. Dieser Bedarf muss im System für jeden Rechenkern vorgehalten werden. Ansonsten ist die Integration eines virtualisierten Steuergerätes nicht möglich, da der Rechenbedarf im aktuellen Demonstratoraufbau nicht auf mehrere Rechenkerne verteilt werden kann.

Der Mehraufwand für den Zugriff auf Peripheriecontroller hängt vom jeweiligen Anwendungsfall ab. Besteht keine Notwendigkeit einer Aufteilung oder dem Schutz von Hardwarecontrollern, können diese direkt an das jeweilige Gastsystem weitergeleitet werden. Dies ist im dargestellten Demonstratorbeispiel zur CAN-Kommunikation der Fall. Somit wird nahezu die native Zugriffsgeschwindigkeiten zur Erreichung des gewünschten Echtzeitverhaltens erzielt.

Im Fall eines Softwarefehlers innerhalb einer VM existiert die Möglichkeit, diese VM neu zu starten, ohne dabei andere Gastsysteme zu stören. Dies wurde innerhalb des Demonstrators im Rahmen eines gesonderten Testszenarios geprüft (siehe *Fallback* in Abbildung B.6). In diesem Fall kann entweder der HV das System in einen sicheren Zustand überführen oder eine andere VM könnte die gewünschte Funktionalität der Vorgänger-VM übernehmen (Hot-Standby-Betrieb).

8.5 Zusammenfassung des Kapitels

Innerhalb dieses Kapitels wurde die Integration von virtualisierten Fahrzeugsoftwarekomponenten auf einer realistischen Hardwareplattform durchgeführt. Zur Konsolidierung von mehreren Fahrzeugsoftwaresystemen repräsentiert der HV eine zusätzliche Softwareschicht, um einzelne Systeme voneinander zu entkoppeln. Der Infineon TriCore AURIX wird bereits in aktuellen Steuergeräteprojekten verwendet und wurde innerhalb dieser Arbeit in vorhergehenden Kapiteln evaluiert (siehe Kapitel 6). Zur Softwareintegration wurden typische Fahrzeugsoftwareapplikationen verwendet.

Das eigens für diese Arbeit erstellte Hochintegrationsszenario beweist die Machbarkeit zur Verwendung eines HVs, um sowohl mehrere AUTOSAR-basierte Softwaresysteme als auch Legacy Code (Displaytreiber) auf einer gemeinsamen Plattform zu konsolidieren. Integrierte Systeme sind dabei durch den HV voneinander isoliert. Darüber hinaus konnte die BSW-Konfiguration vollständig übernommen werden. Das Kommunikationsverhalten einzelner virtualisierter Steuergeräte wurde durch das einfache Hinzufügen von Frontend- und Backend-Treibern nachgebildet. Die verwendeten Fahrzeugsoftwaresysteme wurden somit vollständig in die virtualisierte Umgebung überführt und sind für den realen Fahrzeugbetrieb einsatzfähig.

Durch Laufzeitanalysen wurde gezeigt, dass die Integration einer zusätzlichen HV-Schicht ein Fünftel des Rechenaufwands des virtualisierten Systems verursacht. Ein Großteil dieser Bedarfe wird durch die Schutzmaßnahmen, zur Realisierung nichtunterbrechbarer Codeabschnitte und des HVs zur Änderung der CPU-Priorität verbraucht. Die gemessenen Laufzeiten zur Interruptverarbeitung oder der Zugriff auf einzelne Register bereiten für verwendeten Softwarekomponenten keine Probleme.

Werden allerdings größere Systeme zusammengefasst, muss deren korrekte Funktionsweise und Performanz innerhalb einer virtualisierten Umgebung stets neu beurteilt werden. In Zukunft sollten erweiterte Schutzkonzepte der Hardware eingeführt werden, um den Rechenbedarf einer zusätzlichen Virtualisierungsschicht einzudämmen.

Für sicherheitsrelevante Systeme gilt es zu berücksichtigen, dass ein HV aufgrund seiner zentralen Stellung im Gesamtsystem mit dem höchsten ASIL zu qualifizieren ist, welcher im Steuergerät vorherrscht. Der HV stellt dabei einen *Single Point of Failure* dar und gefährdet im Fehlerfall das Gesamtsystem (siehe Kapitel 6.6.4).

Die in dieser Arbeit dargestellte Methode zur Verlagerung geteilter Hardwaretreiber innerhalb einer gesonderten VM eignet sich für Fahrzeugsoftwaresysteme. Aufgrund des erhöhten Kommunikationsaufwandes ist eine Mischform zwischen einer Treiberintegration innerhalb der DomM und einem DDA-Ansatz für vereinzelte DomUs denkbar (siehe Integration des Displaytreibers), um die geforderten Echtzeiteigenschaften gewährleisten zu können.

Darüber hinaus wurden die hier produzierten Ergebnisse zur Kommunikation zwischen VMs bereits in anderen Bereichen evaluiert. Die Methoden wurden dabei unter Verwendung des Xen Hypervisors auf einer ARM Plattform untersucht [125].

9 Zusammenfassung und Ausblick

9.1 Ergebnisse der Arbeit

Innerhalb dieser Arbeit wurde dargelegt, dass die Paravirtualisierung eingebetteter Fahrzeugsoftwaresysteme als eine Schlüsseltechnologie zur Konsolidierung von Steuergeräten auf einer gemeinsamen Hardwareplattform fungieren kann. Es wurde gezeigt, dass unter Verwendung von aktuell verfügbaren Mikrocontrollern der Automobilindustrie die Portierung von bestehenden Softwaresystemen und der darauffolgende Betrieb innerhalb virtualisierter Softwarecluster möglich ist.

Zur grundsätzlichen Beurteilung der Partitionierbarkeit von Softwarenetzwerken wurde der PageRank Algorithmus an die Bedarfe der Automobilindustrie angeglichen. Die Methoden wurden dabei anhand eines realen Softwarenetzes eines Integrationssteuergerätes evaluiert. Hierbei wurde gezeigt, dass aktuelle Fahrzeuganwendungen stark voneinander abhängen. Insbesondere sicherheitsrelevante Anteile sind hierbei mit einer Vielzahl von benachbarten Softwarekomponenten verbunden und können nur schwer aus bestehenden Softwarestrukturen herausgelöst werden.

Im weiteren Verlauf der Arbeit wurden unterschiedliche Architekturansätze zur Erfüllung nicht-funktionaler Anforderungen (wie funktionale Sicherheit) bewertet. Zur Gegenüberstellung und Entscheidungsfindung der geeignetsten Kernelarchitektur eines monolithischen, Mikrokern-basierten oder Hypervisor-basierten Softwaresystems wurde eigens ein Vorgehensmodell erstellt. Die Evaluation der Methodik ergab für das vorliegende Hochintegrationsszenario, dass zur Konsolidierung sicherheitsrelevanter Software der Hypervisor-basierte Architekturansatz verwendet werden soll.

Daraufhin wurden die Auswirkungen einer zusätzlichen Virtualisierungsschicht im Gesamtsystem einer monolithischen bzw. einer Mikrokern-basierten Lösung gegenübergestellt. Dabei wurde geprüft, in welcher Art und Weise bereits bestehende Kernelarchitekturen nicht-funktionale Anforderungen, unter anderem Portabilität und funktionale Sicherheit, umsetzen können.

Exemplarisch wurde zur Analyse der Kosten einer zusätzlichen Virtualisierungsschicht der Hypervisor der Firma ETAS Ltd. eingesetzt und anhand von Laufzeitmessungen bewertet. Eine Virtualisierung des Systems und die Abschottung einzelner virtueller Maschinen durch den Hypervisor wurden erreicht. Analysen ergaben allerdings, dass beim Einsatz aktueller Mikrocontrollertechnologien besonders zeitliche Einbußen (Virtualisierung der Interrupts) in Kauf genommen werden müssen. Dies wird durch fehlende Virtualisierungserweiterungen der Hardware begründet, welche somit durch

Laufzeit-intensive Softwareerweiterungen zu ersetzen sind. Hardware-gestützte Vorgänge, wie zur Weiterleitung von Interrupts, werden dadurch verlangsamt.

Für leistungsschwächere Mikrocontroller mit harten Echtzeitanforderungen und zugleich monetären Einschränkungen ist somit zusätzliche Unterstützung der Hardware notwendig, um effizientere und flexiblere Integrationslösungen zu ermöglichen. Die Prozessorarchitektur soll hierbei zusätzliche Berechtigungsstufen für einen Hypervisor zur Verfügung stellen. Dabei müssen Hardwareressourcen, wie Kommunikationscontroller, gleichzeitig von mehreren virtuellen Maschinen verwendet werden können.

Um das bereits existierende Kommunikationsverhalten konsolidierter Steuergeräte über verwendete Feldbusse in einer virtualisierten Umgebung nachstellen zu können, wurden virtuelle Maschinen unter Verwendung eines generischen, asynchronen Kommunikationskanals miteinander verbunden. Das Konzept wurde eigens im Rahmen dieser Arbeit implementiert und Laufzeitmessungen durchgeführt. Dabei wurde deutlich, dass sowohl aufgrund häufiger Kopieroperationen als auch unter Verwendung der noch leistungsschwachen Hardware hohe Übertragungsraten nicht erreichbar sind.

Es wurde weiterhin festgestellt, dass die Integration einer gemeinsamen AUTOSAR Treiberschicht eine besondere Rolle zur Konsolidierung sicherheitsrelevanter Fahrzeugsoftware innerhalb einer Integrationsplattform spielt. Hierfür wurde eine Methodik vorgeschlagen und evaluiert, um gemeinsam genutzte Hardwaretreiber von betreffenden virtuellen Maschinen zu trennen. Dabei wird die Treiberschicht AUTOSAR-basierter Softwaresysteme in eine isolierte, privilegierte virtuelle Maschine verlagert. Der Signalaustausch erfolgt durch den bereits erwähnten Kommunikationskanal.

Zur Finalisierung der Arbeit und Evaluation der erarbeiteten Konzepte wurde Fahrzeugsoftware innerhalb eines Hochintegrationsdemonstrators auf einer gemeinsamen Hardwareplattform zusammengefasst. Als Mikrocontroller wurde hierfür der Infineon TriCore AURIX ausgewählt, welcher mehrfach in aktuellen Fahrzeugderivaten verbaut ist. Laufzeitmessungen haben gezeigt, dass der Einsatz eines Hypervisors unter reellen Bedingungen ein Fünftel des Rechenbedarfs verursacht. Der erhöhte Laufzeitbedarf, die verlangsamte Verarbeitung von Interrupts und der erschwerte Zugriff auf geteilte Hardwareressourcen verdeutlichen die Notwendigkeit zur Einführung unterstützender Virtualisierungserweiterungen für zukünftige Mikrocontroller.

Speziell für sicherheitsrelevante Systeme stellen die Abschottungsmöglichkeiten zwischen virtuellen Maschinen eine verlässliche Methode dar, um Rückwirkungsfreiheit zwischen Softwaresystemen mit unterschiedlichen Sicherheitseinstufungen zu gewährleisten. Diese Isolationsmechanismen werden durch einen qualifizierten Hypervisor zur Verfügung gestellt. Dieser wird im Kontext der Trusted Computing Base betrieben und muss daher nach der höchsten integrierten Sicherheitseinstufung des Steuergeräts qualifiziert werden.

Einerseits erhöht die zusätzliche Softwareschicht eines Hypervisors zur Abstraktion virtueller Maschinen die Komplexität eines Systems, andererseits ermöglicht diese Schicht mehrere Freiheitsgrade zur Entkoppelung von Fahrzeugsoftwaresystemen untereinander. In jedem Fall bietet ein Hypervisor zusätzliche Isolationsmechanismen für sicherheitsrelevante Anwendungen, was einen Mehrwert für aktuelle Fahrzeugsysteme darstellt. Die Entscheidung über den Einsatz eines Hypervisors muss somit immer

projekt- und problembezogen getroffen werden. Es muss dabei der Mehrwert der Softwareisolation, den ein virtualisiertes System mit sich bringt, mit den Nachteilen des zusätzlichen Speicherbedarfs und dem erhöhten Laufzeitkosten abgewogen werden.

9.2 Weiterführende Arbeiten

Im Laufe dieser Arbeit entstanden diverse Ideen und Anregungen, welche jedoch den Umfang dieser Arbeit übersteigen. Dabei wurden auch interdisziplinäre Schnittstellen zu verwandten Fachgebieten aufgedeckt.

Der graphenbasierte Ansatz zur Analyse der Abhängigkeiten innerhalb sicherheitsrelevanter Steuergeräte unter Verwendung von Methoden zur Analyse sozialer Netzwerke könnte durch Algorithmen zur optimalen Partitionierung von Graphen erweitert werden. Diese repräsentieren einen gesonderten Teilbereich innerhalb der Graphentheorie. Methoden zur Partitionierung von Mehrkernsystemen sind hierbei bereits bekannt und könnten auf Fahrzeugsoftwaresysteme portiert und angewendet werden.

Bezüglich des eingesetzten Hypervisors von ETAS Ltd. könnte ein direkter Folgeschritt die Analyse der bestehenden Lösung auf dem nachfolgenden Mikrocontroller des Infineon TriCore AURIX sein. Hierbei sollte die Skalierbarkeit des aktuellen Hypervisors innerhalb eines Szenarios mit mehr als drei Rechenkernen geprüft werden.

Weiterhin könnte der Hypervisor erweitert werden, um auch mehrere virtuelle Maschinen pro Rechenkern verwalten zu können. Dies ermöglicht die volle Auslastung eines Rechenkerns, da mehrere kleinere virtualisierte Systeme auf einem Rechenkern zusammengefasst werden könnten. Aus evolutionärer Sicht stellt die Portierung des verwendeten Hypervisors (RTA-HV) auf eine fremdartige Hardware den nächsten logischen Schritt dar. Hierfür kommt eine ARM bzw. PowerPC Prozessorarchitektur in Frage. Es gilt zu prüfen, ob das Sicherheitsmodell und die in dieser Arbeit dargestellten Vorgehensweisen allgemeingültig auf zukünftigen Plattformen einsetzbar sind.

Das Konzept zur Verlagerung der AUTOSAR Treiberschicht ist bisher noch nicht im AUTOSAR Standard erwähnt und berücksichtigt. Zudem sind virtualisierte Systeme im aktuellen AUTOSAR Standard nicht vorgesehen. Eine Erweiterung der Spezifikation zur Einführung von Schnittstellen zwischen virtualisierten Systemen könnte dabei die Migration von virtualisierten Plattformen durch einen standardisierten Integrationsprozess erleichtern.

Der realisierte Demonstratoraufbau konsolidiert eine begrenzte Anzahl von Fahrzeugfunktionen. Weitere Laufzeitmessungen könnten an größeren konsolidierten Systemen durchgeführt werden, welche Fahrzeugfunktionen unterschiedlicher Domänen zusammenfassen. Hierbei kommen besonders Funktionalitäten der Domäne Infotainment und zukünftige Anwendungen für Fahrerassistenzsysteme in Frage.

Es wurde weiterhin gezeigt, dass geteilte Hardwareressourcen erhöhten Integrationsaufwand und Laufzeitkosten bedingen. Daher sollten selbst-virtualisierende Hardwarecontroller (wie CAN oder Ethernet [79, 119]) in Verbindung mit einer automotivetauglichen Hypervisorlösung untersucht werden [80].

Weiterhin sollte geprüft werden, ob der Einsatz quelloffener Systeme für diesen Anwendungsfall tauglich ist und den Ansprüchen der Automobilindustrie gerecht wird.

Der Fokus könnte auf Systeme wie Xen oder XtratuM gesetzt werden, welche bereits für ARM Architekturen portiert wurden. Die Konzepte zur Kommunikation von Fahrzeugsoftware zwischen virtuellen Maschinen wurden bereits über den Rahmen dieser Arbeit hinaus untersucht [125].

A Abkürzungen

ASIL Automotive Safety Integrity Level 11
AUTOSAR AUTomotive Open System ARchitecture 3
BISR Begin Interrupt Service Routine 98
BIV Base Interrupt Vector Table Pointer 13
BSW Basic Software 10
BswM Basic Software Mode Manager 76
BTV Base Trap Vector Table Pointer 13
CAN Controller Area Network 14
CAT1 Category 1 ISR 98
CAT2 Category 2 ISR 98
CCF Common Cause Failure 21
CCPN CPU Priority Number 98
CF Cascading Failures 21
CMF Common Mode Failure 21
CPR Code Protection Range 123
CRC Cyclic Redundancy Check 79
CSA Context Save Area 93
DCU Domain Controller Unit 17
DDA Direct Device Assignment 92
DLR Domänenleitrechner 17
DMA Direct Memory Access 138
DomM Master Domain 127
DomU User Domain 127
DPR Data Protection Range 123
E/E elektrisch und elektronisch 3
ECU Electronic Control Unit 12
EcuM ECU State Manager 76

FCM Factor-Criteria-Metrics 43
FMEA Fehlermöglichkeits- und -einflussanalyse 45
FP Fehlerpotential 52
HITS Hypertext-Induced Topic Selection 29
HV Hypervisor 4
ICR Interrupt Control Register 90
IOC Inter-OS-Application Communicator 76
ISA Instruction Set Architecture 8
ISR Interrupt Service Routine 78
IvmcVDE Inter-Virtual-Machine-Communication VDE 129
IVT Interrupt Vector Table 92
KA Kernelarchitektur 43
LMU Local Memory Unit 13
LSSI Large Scale Software Integration 18
M Metrik 52
MCAL Microcontroller Abstraction Layer 10
MCU Microcontroller Unit 110
MFCR Move From Core Register 13
MMU Memory Management Unit 65
MPU Memory Protection Unit 13
MTCR Move To Core Register 13
MTTF Mean Time To Failure 52
NFE nicht-funktionale Eigenschaft 43
OEM Original Equipment Manufacturer 15
OS Operating System 11
PCI Protocol Control Information 131
PIPN Pending Interrupt Priority Number 98
PRS Protection Register Set 105
PSW Program Status Word 13
QA Qualitätsattribut 47
RFE Return From Exception 100
RTE Run Time Environment 10
SAT Source Address Table 133
SU Supervisor 13

SUB Substeuergerät 17
SWC Software Component 10
SwcRank Software Component Rank 27
TCB Trusted Computing Base 7
TM Technische Maßnahme 52
VDE Virtual Device Emulator 90
vECU virtuelles Steuergerät 94
VFB Virtual Function Bus 10
VLSI Very Large Scale Integration 18
VM virtuelle Maschine 4
VMI Virtual Machine Interface 9
WCET Worst Case Execution Time 99

B Abbildungen

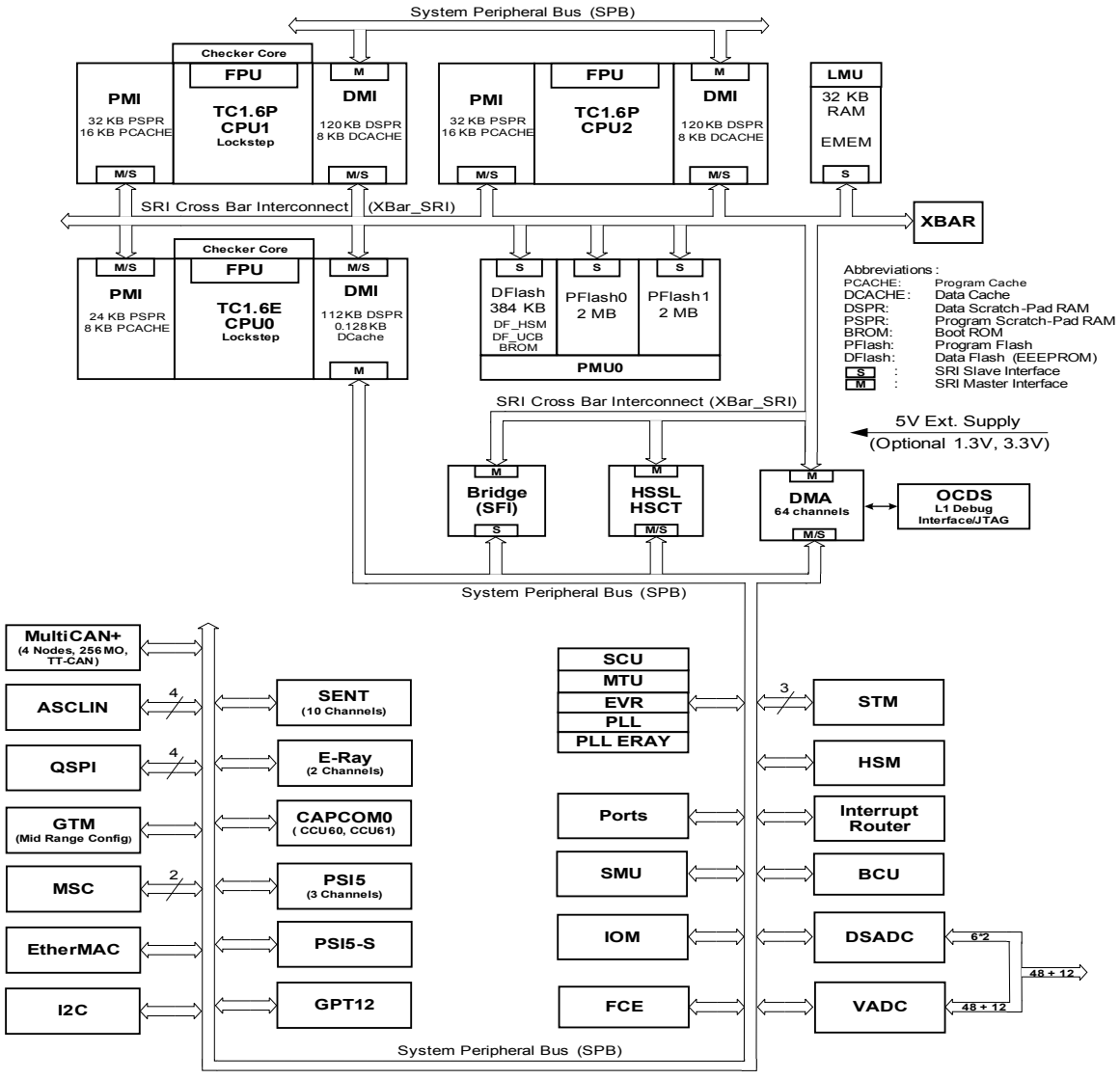


Abbildung B.1: Hardwarearchitektur des Infineon TriCore AURIX TC27x [4].

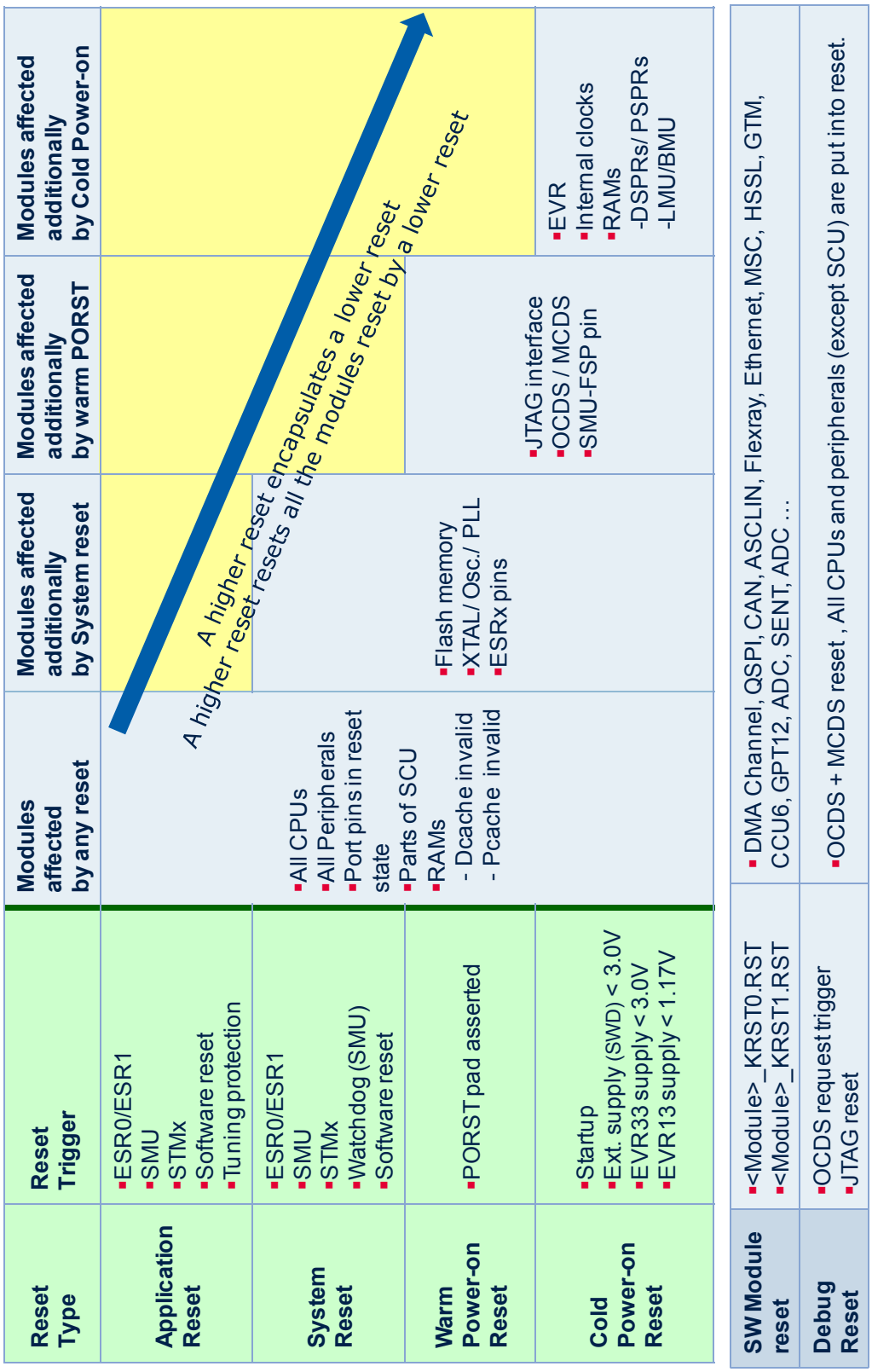


Abbildung B.2: Blockdiagramm zum Startvorgang des Infineon TriCore AURIX Mikrocontrollers [5].

Abbildung B.3: Übersicht aller Teilgebiete der ISO 26262 [89].

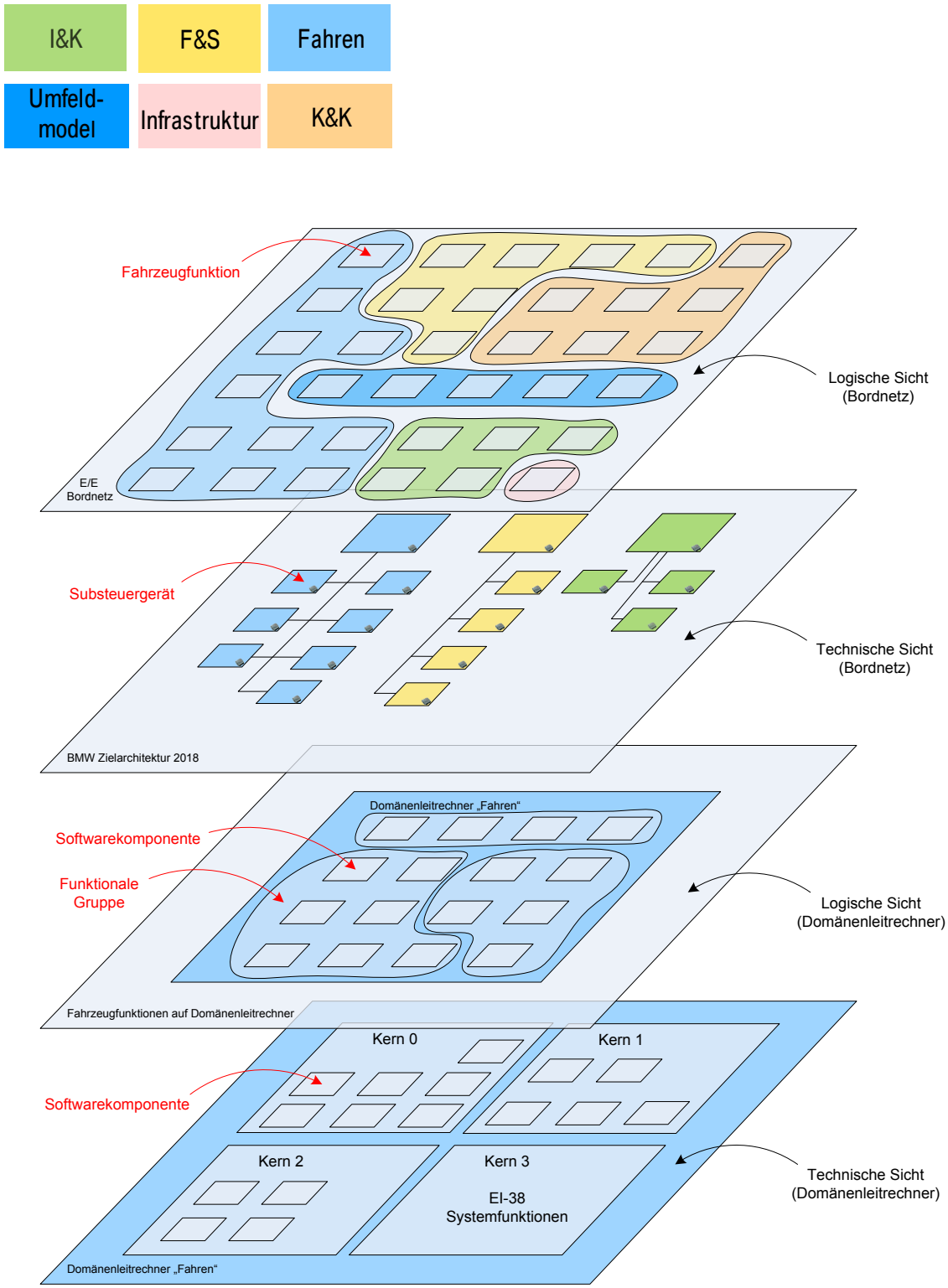


Abbildung B.4: Schichtenmodell zur Partitionierung und Verteilung von Fahrzeugfunktionen auf Domänenleitrechner, ausgehend von einer logischen Bordnetzarchitektur.

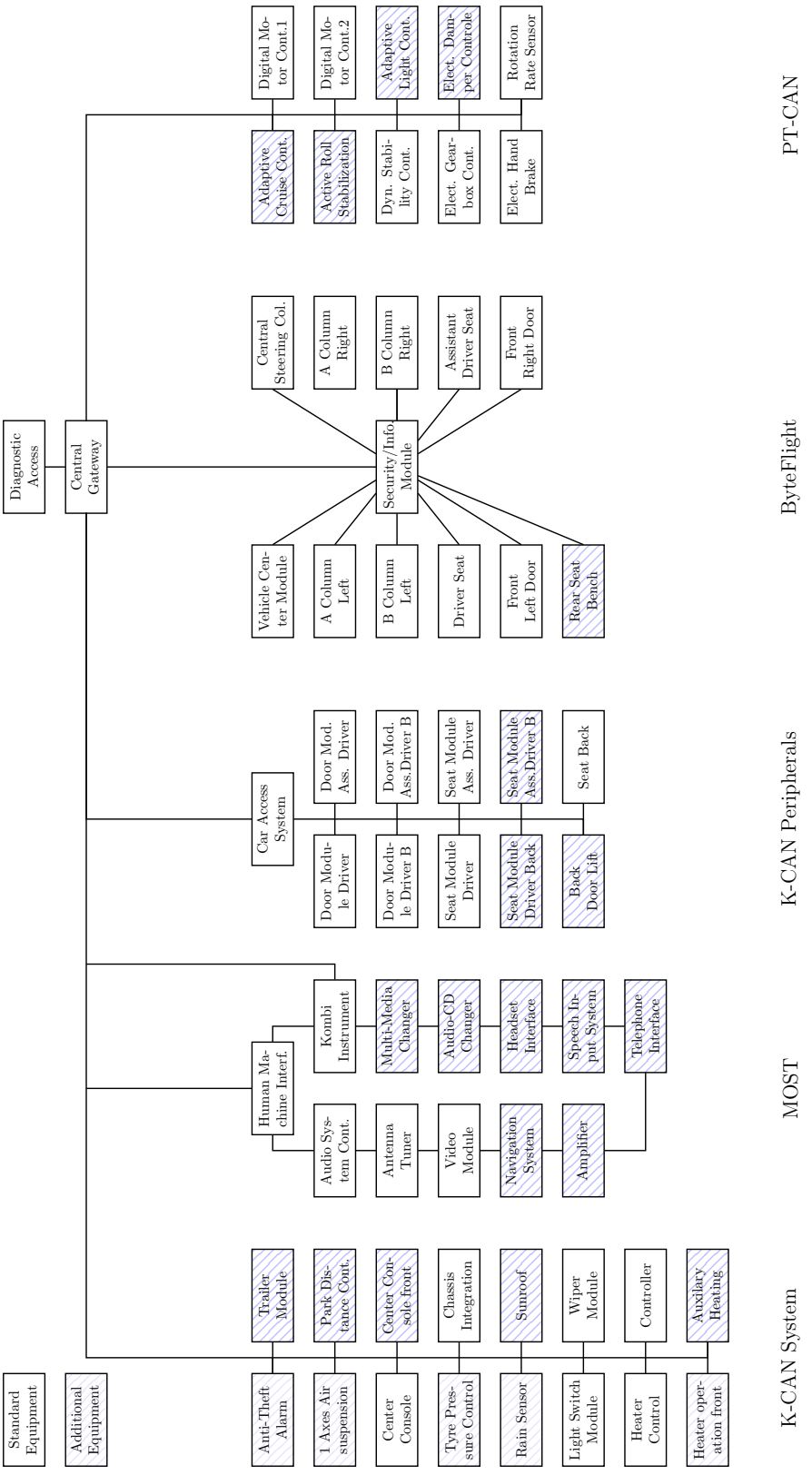


Abbildung B.5: E/E-Systemarchitektur verbundener Steuergeräte am Beispiel von Bordnetz-2000 der BMW AG [121].

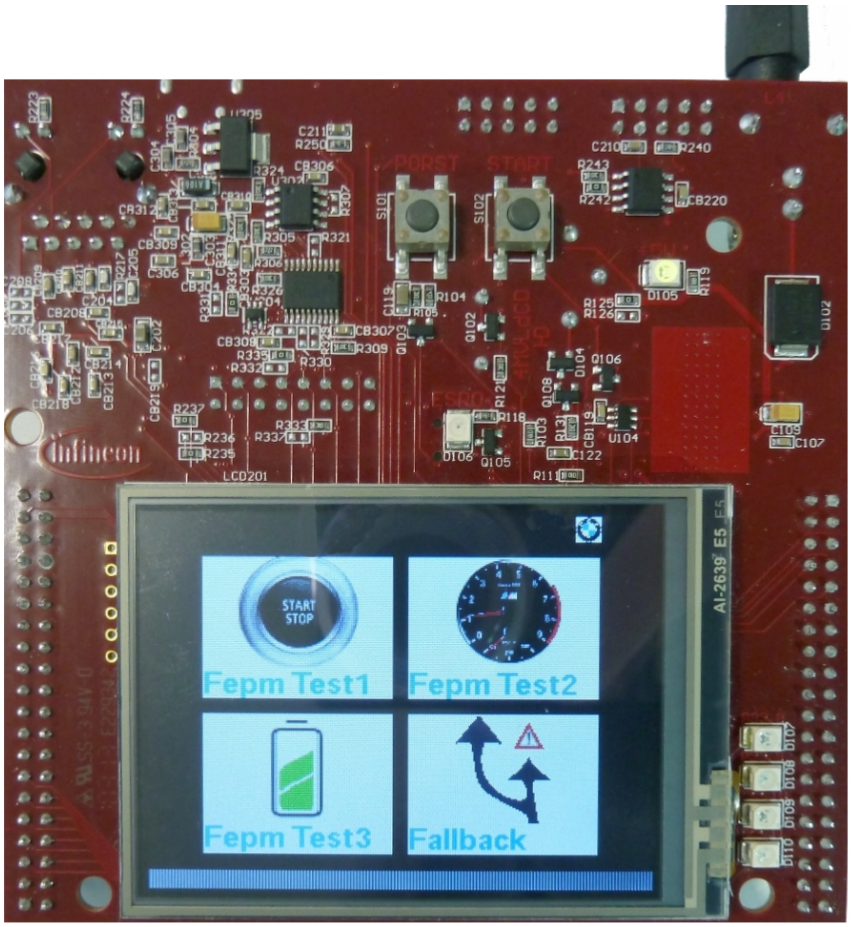


Abbildung B.6: Hochintegrationsdemonstrator der BMW AG erstellt im Rahmen des ARAMiS BMBF Forschungsprojektes auf dem Infineon TriCore AURIX TFT Application Kit [124].

Abbildungsverzeichnis

1.1 Vergleich der Rechenleistungen von Steuergeräten der Automobilindustrie mit der Rechenkapazitätssteigerung innerhalb der Consumer-Elektronik am Beispiel des Entwicklungsverlaufs von Kundenfunktionen der BMW AG [58, 128, 141]. 2

2.1 Kernelaufbau eines mikrokern-basierten bzw. monolithischen Betriebssystems [145]. 8
2.2 Klassifizierung möglicher Hypervisorarchitekturen und -typen [145]. . . 10
2.3 Softwareschichten der AUTOSAR Systemarchitektur [15]. 11
2.4 ASIL Dekomposition laut ISO 26262 [89], zur Aufspaltung hoher Sicherheitseinstufungen in mehrere niedere Einstufungen. 12

3.1 Darstellung einer möglichen Domänen-orientierten Architekturauslegung von Integrationssteuergeräten bzw. Domänenleitrechnern und einzelnen SUB-Steuergeräten [123]. 18
3.2 Logische Segmentierung eines Bordnetzes in funktionale Cluster, zur Verteilung von Vehikelfunktionen auf Domänen- oder Substeuergeräte [121]. 19
3.3 Fehlerfortpflanzung zwischen qualifizierten und nicht-qualifizierten Komponenten. 21
3.4 Partitionierung von Integrationssteuergeräten. 23
3.5 Szenarien zur Verlagerung und Partitionierung auf Bordnetzebene. . . . 24

4.1 Vermaschung sicherheitsrelevanter Software (grün: trusted) und unqualifizierter Software (rot: untrusted) innerhalb der AUTOSAR BSW [152]. 28
4.2 Beispielhafter Graph des Software Component Ranks, wobei die Gewichtung der Knoten durch die Farbintensität ausgedrückt ist. 36
4.3 Darstellung des Softwarenetzes eines Integrationssteuergerätes inklusive deren sicherheitsrelevanter Softwarekomponenten [120]. 38

5.1 Aufschlüsselung der Systemzuverlässigkeit in Attribute, Gefahren und Maßnahmen nach Laprie [101]. 45
5.2 Hierarchische Ebenen des Entscheidungsbaumes. 52
5.3 Fehlerwirkungskette im Zusammenhang von Fehlerpotentialen und technischen Maßnahmen. 53
5.4 Exemplarischer Aufbau einer gewichteten Baumstruktur. 55

5.5 Zusammenführung redundanter Teilbäume. 57
5.6 Maßnahmen zum Umgang mit Fehlern im Rahmen der funktionalen Sicherheit, dargestellt als Baumstruktur. 61
5.7 Detailausführung zur Fehlervermeidung von Abbildung 5.6. 63
5.8 Detailausführung der Fehlererkennung zur Designphase und Laufzeit (siehe Abbildung 5.6). 66

6.1 Intrusive und Non-Intrusive Softwareintegration. 74
6.2 AUTOSAR Multicore Architektur und deren Softwarepartitionen [123]. 76
6.3 Logische und technische Sicht zur Kapselung von Systemelementen [123]. 79
6.4 Task-Allokation innerhalb eines Mehrkernsystems [16, 123]. 80
6.5 Grundidee einer Mikrokern-basierten AUTOSAR Schichtenarchitektur [29]. 82
6.6 Partitionierung von Fahrzeugsoftwaresystemen unter Verwendung eines Typ-1 Hypervisors. 84
6.7 Integration sicherheitsrelevanter Software unter Verwendung eines Hypervisors. 86
6.8 Grundfunktionen des Hypervisors. 90
6.9 Schichtenarchitektur eines virtualisierten Systems unter Verwendung der Infineon TriCore AURIX Prozessorarchitektur. 91
6.10 Ausnahmebehandlungen des paravirtualisierten Systemaufbaus [92]. . . 92
6.11 Systemzustände des Hypervisors und dessen virtueller Maschinen. . . . 93
6.12 Prozessablauf zur Weiterleitung von Interrupts der Peripheriecontroller gerichtet an eine virtuelle Maschine [122]. 97
6.13 Ablauf zur Interruptverarbeitung innerhalb virtualisierter Plattformen [64, 92]. 99
6.14 Interruptverarbeitung und deren Kosten innerhalb virtualisierter Umgebungen [64, 92]. 99
6.15 Ablauf eines Kontextwechsels zur Aktivierung von Tasks innerhalb virtualisierter Umgebungen [64, 92]. 100
6.16 Aktivierungskosten von Tasks und deren zeitliche Aufwände [64, 92]. . . 101
6.17 Zeitlicher Aufwand zum Wechsel zwischen Partitionen im Kernel- und User-Modus [64, 92]. 102
6.18 Gegenüberstellung zur Bewertung der Architekturprinzipien: Monolith, Mikrokern und Hypervisor. 112

7.1 Fahrzeugbus-übergreifender Vernetzungsgrad zwischen farbig gekennzeichneten Softwarekomponenten von drei Integrationssteuergeräten [126]. 120
7.2 Ansätze zur Verlagerung der AUTOSAR Treiberschicht [126]. 125
7.3 Konsolidierung einzelner Steuergeräte und Abbildung virtualisierte Busstrukturen [126]. 129
7.4 Kommunikationsmechanismus zur Interaktion zwischen virtuellen Maschinen unter Verwendung von Frontend- bzw. Backend-Treibern [126]. 130

7.5 Domänenübergreifende Übertragung von Ethernet Frames zwischen virtuellen Maschinen. 136
7.6 Domänenübergreifende Übertragung von CAN-Frames zwischen virtuellen Maschinen. 137
7.7 Maximale Datenrate zwischen virtuellen Maschinen [113, 126]. 139
7.8 Gemessene Datenraten zwischen VMs unter Verwendung des IvmcVDE [68, 126]. 139

8.1 Ansätze zur Integration eines Schnittstellentreibers zum Nachrichtenaustausch mit einem Kommunikationskanal innerhalb des Hypervisors. 146
8.2 Softwarearchitektur des Hochintegrationsdemonstrators zur Konsolidierung von unterschiedlichen Fahrzeugsoftwarepartitionen. 149
8.3 Kommunikationsmöglichkeiten zwischen virtuellen Maschinen innerhalb des Demonstratoraufbaus. 150
8.4 Grundlegender Startup-Prozess zur Hardwareinitialisierung eines AUTOSAR-basierten Systems. 152
8.5 Initialisierung virtualisierte Steuergeräte unter Verwendung des MCU-Moduls. 152
8.6 Auslastung der Rechenkerne in Partition 0 und 1 innerhalb des Demonstratoraufbaus gemessen über einen Zeitraum von 60 Sekunden [7, 64]. 154
8.7 Funktionsbeanspruchung durch Registerzugriffe des Hypervisors in Partition 0 und 1 innerhalb des Demonstratoraufbaus, gemessen über einen Zeitraum von 60 Sekunden [7, 64]. 155

B.1 Hardwarearchitektur des Infineon TriCore AURIX TC27x [4]. 167
B.2 Blockdiagramm zum Startvorgang des Infineon TriCore AURIX Mikrocontrollers [5]. 168
B.3 Übersicht aller Teilgebiete der ISO 26262 [89]. 169
B.4 Schichtenmodell zur Partitionierung und Verteilung von Fahrzeugfunktionen auf Domänenleitrechner, ausgehend von einer logischen Bordnetzarchitektur. 170
B.5 E/E-Systemarchitektur verbundener Steuergeräte am Beispiel von Bordnetz-2000 der BMW AG [121]. 171
B.6 Hochintegrationsdemonstrator der BMW AG erstellt im Rahmen des ARAMiS BMBF Forschungsprojektes auf dem Infineon TriCore AURIX TFT Application Kit [124]. 172

Tabellenverzeichnis

4.1 Vergleich der Wichtigkeiten bzw. Zentralität einzelner Softwareknoten. 36

5.1 Resultate zur Evaluation der Methodik hinsichtlich funktionaler Sicherheit (siehe Abbildung 5.7, 5.6 und 5.8). 69

6.1 Gegenüberstellung von Voll- und Paravirtualisierung für eingebettete Fahrzeugsysteme. 85
6.2 Messergebnisse zur Analyse des zeitlichen Mehraufwands innerhalb des paravirtualisierten Systems [122]. 96
6.3 Unterstützung des Speicherschutzes durch den Hypervisor. 104
6.4 Unterstützung des zeitlichen Schutzes durch den Hypervisor. 106
6.5 Unterstützung des Informationsschutzes durch den Hypervisor. 107

7.1 Nachrichtenaufbau eines CAN bzw. Ethernet Frames. 131
7.2 Exemplarische Routingtabelle zur Weiterleitung von CAN-Frames. . . . 134

Literaturverzeichnis

[1] Daniel Adam, Joachim Froeschl, Uwe Baumgarten, Andreas Herkersdorf, and Hans-Georg Herzog. Cyber Organic System-Model – New Approach for Automotives System Design. In Carsten Hoff and Ottmar Sirch, editors, *Elektrik/-Elektronik in Hybrid- und Elektrofahrzeugen VI*, pages 161 – 186. expert-Verl., April 2015. ISBN: 978-3-8169-3311-3.

[2] Infineon Technologies AG. *TriCore TC1.6P & TC1.6E Core Architecture 32-bit Unified Processor Core*. Infineon Technologies AG, Neubiberg, Deutschland, 1 edition, February 2014.

[3] Infineon Technologies AG. *TriCore TC1.6P & TC1.6E Instruction Set Architecture 32-bit Unified Processor Core*. Infineon Technologies AG, Neubiberg, Deutschland, 2 edition, February 2014.

[4] Infineon Technologies AG. *AURIX TC27x 32-Bit Single-Chip Microcontroller*. Infineon Technologies AG, Neubiberg, Deutschland, v1.3.1 edition, February 2014.

[5] Infineon Technologies AG. *TriCore AURIX Family - TC27x - Start-up and Initialization*. Infineon Technologies AG, Neubiberg, Deutschland, 1 edition, February 2014.

[6] M. Aichouch, J.-C. Prevotet, and F. Nouvel. Evaluation of the Overheads and Latencies of a Virtualized RTOS. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 81–84, June 2013. doi: 10.1109/SIES.2013.6601475.

[7] ARAMiS Consortium. *Automotive, Railway and Avionics Multicore Systems*. Technische Universität Karlsruhe, March 2015.

[8] C. Aussagues, D. Chabrol, V. David, D. Roux, N. Willey, A. Tournadre, and M. Graniou. Pharos, a multicore os ready for safety-related automotive systems: results and future prospects. In *The Embedded Real-Time Software and Systems (ERTS2)*, 2010.

[9] AUTOSAR. Guide to Multi Core Systems, 2013. Version 4.1.1 – Final.

[10] AUTOSAR. Specification of ECU State Manager. Version 4.2.1, October 2014.

[11] AUTOSAR. Specification of ECU Configuration. Version 4.2.1, 2014.

[12] AUTOSAR. Software Component Template. Version 4.2.1, 2014.

[13] AUTOSAR. Virtual Functional Bus. Version 4.2.1, October 2014.

[14] AUTOSAR. Specification of SW-C End-to-End Communication Protection Library. Version 4.2.1, 2014.

[15] AUTOSAR. Layered Software Architecture. Version 4.2.1, October 2014.

[16] AUTOSAR. Specification of Operating System. Version 4.2.1, October 2014.

[17] AUTOSAR. Explanation of Application Interfaces of the Chassis Domain. Version 4.2.1, 2014.

[18] AUTOSAR. Explanation of Application Interfaces of the Powertrain Domain. Version 4.2.1, 2014.

[19] AUTOSAR. Explanation of Interrupt Handling within AUTOSAR. Version 4.2.1, October 2014.

[20] AUTOSAR. Guide to Multi Core Systems, 2014. Version 4.2.1.

[21] AUTOSAR. Main Requirements. Version 4.2.1, October 2014.

[22] AUTOSAR. Requirements on Software Component Template. Version 4.2.1, October 2014.

[23] AUTOSAR. Specification of CAN Network Management. Version 4.2.1, 2014.

[24] AUTOSAR. Specification of Diagnostic Event Manager. Version 4.2.1, October 2014.

[25] AUTOSAR. Specification of MCU Driver. Version 4.2.1, October 2014.

[26] AUTOSAR. System Template. Version 4.2.1, October 2014.

[27] AUTOSAR. Specification of Timing Extensions, March 2014. Version 4.2.1.

[28] AUTOSAR. Methodology. Version 4.2.1, October 2014.

[29] AUTOSAR Administration. AUTomotive Open System ARchitecture. http://www.autosar.org, October 2014. v4.2.1.

[30] AUTOSAR Administration. AUTomotive Open System ARchitecture. http://www.autosar.org, 2014.

[31] H. Balzert and C. Ebert. *Lehrbuch der Softwaretechnik: Softwaremanagement*. Lehrbuch der Software-Technik. Spektrum Akademischer Verlag, 2008. ISBN 9783827411617.

[32] H. Balzert, H. Balzert, R. Koschke, U. Lämmel, P. Liggesmeyer, and J. Quante. *Lehrbuch Der Softwaretechnik: Basiskonzepte Und Requirements Engineering*. Lehrbuch der Software-Technik. Spektrum Akademischer Verlag, 2009. ISBN 9783827422477.

[33] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: 10.1145/945445.945462.

[34] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003. ISSN 0163-5980. doi: 10.1145/1165389.945462.

[35] Andreas Barthels, Hans-Ulrich Michel, and Gregor Walla. Jedes Watt zählt - Intelligentes Energie- und Leistungs-Management für die Autos von morgen. *Elektronik automotive*, 05:24–28, May 2012.

[36] J. Bengtsson and B. Svensson. Manycore performance analysis using timed configuration graphs. In *International Symposium on Systems, Architectures, Modeling, and Simulation (SAMOS 2009)*, pages 108–117, July 2009. doi: 10.1109/ICSAMOS.2009.5289221.

[37] ESA Inquiry Board. Ariane 5 Flight 501 Failure. Technical report, European Space Agency, July 1996.

[38] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[39] Barry W. Boehm, John R. Brown, and Hans Kaspar. *Characteristics of Software Quality. Vol. 1.* TRW series of software technology. North-Holland Publishing, Amsterdam, New York, 1978. ISBN 0-444-85105-4.

[40] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall Advances in Computing Science & Technology Series. Pearson Education, 1981. ISBN 9780138221225.

[41] Michael Bohn, Jörn Schneider, Christian Eltges, and Robert Rößger. Migration von AUTOSAR-basierten Echtzeitanwendungen auf Multicore-Systeme. In *4. Workshop „Entwicklung zuverlässiger Software-Systeme“*, pages 5–11, 2011.

[42] Phillip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.

[43] Phillip Bonacich. Some Unique Properties of Eigenvector Centrality. *Social Networks*, 29(4):555–564, October 2007. ISSN 03788733. doi: 10.1016/j.socnet.2007.04.002.

[44] Phillip Bonacich and Paulette Lloyd. Eigenvector-like Measures of Centrality for Asymmetric Relations. *Social Networks*, 23(3):191–201, July 2001. doi: 10.1016/S0378-8733(01)00038-7.

[45] J. Börcsök. *Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme*. VDE-Verlag, 2011. ISBN 9783800733057.

[46] Josef Börcsök. *Elektronische Sicherheitssysteme, Hardwarekonzepte, Modelle und Berechnung*. Hthig, 2., berarb. aufl. edition, 2007. ISBN 978-3-7785-4021-3. Josef Brcsk.

[47] Simon Brewerton and Rolf Schneider. Hardware Based Paravirtualization: Simplifying the Co-Hosting of Legacy Code for Mixed Criticality Applications. In *SAE 2013 World Congress & Exhibition*, April 2013. doi: 10.4271/2013-01-0186.

[48] Simon P. Brewerton, Natalia Willey, Swapnil Gandhi, Thorsten Rosenthal, Claus Stellwag, and Matthieu Lemerre. Demonstration of Automotive Steering Column Lock using Multicore AutoSAR® Operating System. In *SAE International*, January 2012.

[49] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

[50] Manfred Broy. Challenges in Automotive Software Engineering. In *Proc. ACM 28th International Conference on Software Engineering (ICSE 2006)*, pages 33–42, Shanghai, China, May 2006.

[51] G. Buttazzo, E. Bini, and Yifan Wu. Partitioning Real-Time Applications Over Multicore Reservations. *Industrial Informatics, IEEE Transactions on*, 7(2): 302–315, May 2011. ISSN 1551-3203. doi: 10.1109/TII.2011.2123902.

[52] E. Carrascosa, J. Coronel, M. Masmano, P. Balbastre, and A. Crespo. Xtratum hypervisor redesign for leon4 multicore processor. *SIGBED Rev.*, 11(2):27–31, September 2014. ISSN 1551-3688. doi: 10.1145/2668138.2668142.

[53] Eurocontrol Experimental Centre. Revisiting the Swiss Cheese Model of Accidents. In *European Organisation for the Safety of Air Navigation (Eurocontrol)*, volume EEC Note No. 13/06, October 2006.

[54] Albrecht Mayer Claus Stellwag, Jens Harnisch. Interference in Time, Analysis and Optimization Options on a Multi-Core Microcontroller. In *Embedded World Converence 2014*, February 2014.

[55] DIN EN 50126. Railway applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 2000-03.

[56] Radio Technical Commission for Aeronautics (RTCA). Do-178b: Software considerations in airborne systems and equipment certification. Technical report, Radio Technical Commission for Aeronautics (RTCA), 1982.

[57] L.C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40:35–41, 1977.

[58] Elmar Frickenstein. Die it ist im fahrzeug angekommen. CEBIT 2011, February 2011.

[59] Martin Fuchs, Patrick Scheer, and Andreas Grzemba. Selektiver Teilnetzbetrieb im Fahrzeug: Eine Realisierung für den CAN-Bus und Adaption auf andere Bussysteme. In *AmE 2010 - Automotive meets Electronics - GMM-Fachtagung*, pages 7–11. VDE-Verlag, April 2010.

[60] Simon Fürst. Challenges in the Design of Automotive Software. In *Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 256 – 258, March 2010. doi: 10.1109/DATE.2010.5457201.

[61] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, January 1995. ISBN 0201633612.

[62] Robert P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, Cambridge, MA, February 1972.

[63] Robert B. Grady and Deborah L. Caswell. *Software Metrics: Establishing a Company-wide Program*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 0-13-821844-7.

[64] Andreas Graf. Achieving fault and performance isolation on the aurix mpsoc using virtualization. Master's thesis, Technische Universität München, Institute for Integrated Systems, April 2014.

[65] S. Groesbrink. Virtual machine migration as a fault tolerance technique for embedded real-time systems. In *Eighth IEEE International Conference on Software Security and Reliability-Companion (SERE-C)*, pages 7–12, June 2014. doi: 10.1109/SERE-C.2014.16.

[66] S. Groesbrink, L. Almeida, M. de Sousa, and S. M. Petters. Towards Certifiable Adaptive Reservations for Hypervisor-based Virtualization. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, Germany, April 2014.

[67] Stefan Groesbrink and Luis Almeida. A Criticality-Aware Mapping of Real-time Virtual to Multicore Processors. In *19th IEEE International Conference on Emerging Technologies and Factory Automation*, 2014.

[68] Maximilian Güntner. Virtualization of Time Critical Communication Interfaces. Bachelor's thesis, University of Applied Sciences Kempten, September 2014.

[69] Georg Gut, Christian Allmann, Markus Schurius, and Karsten Schmidt. Reduction of Electronic Control Units in Electric Vehicles Using Multicore Technology. In Victor Pankratius and Michael Philippsen, editors, *Multicore Software Engineering, Performance, and Tools*, volume 7303 of *Lecture Notes in Computer Science*, pages 90–93. Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-31201-4.

[70] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with TrustRank. Technical Report 2004-17, Stanford InfoLab, March 2004.

[71] Anthony Hall and Roderick Chapman. Correctness by Construction: Developing a Commercial Secure System. *IEEE Software*, Jan/Feb:18–25, 2002.

[72] Mian M. Hamayun, Alexander Spyridakis, and Daniel S. Raho. Towards Hard Real-Time Control and Infotainment Applications in Automotive Platforms. In *Proceedings of the* 10^{th} *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2014)*, July 2014.

[73] David Haworth. An AUTOSAR-compatible microkernel for systems with safety-relevant components. *Informatik aktuell*, Herausforderungen durch Echtzeitbetrieb:11–20, 2012.

[74] Gernot Heiser. The Role of Virtualization in Embedded Systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, IIES '08, pages 11–16, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-126-2. doi: 10.1145/1435458.1435461.

[75] Gernot Heiser. Hypervisors for consumer electronics. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5, January 2009. doi: 10.1109/CCNC.2009.4784922.

[76] Gernot Heiser and Ben Leslie. The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors. In *Proceedings of the 1st Asia-Pacific Workshop on Systems (APSys)*, pages 19–24, New Delhi, India, August 2010.

[77] C. Herber, A. Richter, T. Wild, and A. Herkersdorf. A Network Virtualization Approach for Performance Isolation in Controller Area Network (CAN). In *The 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[78] C. Herber, A. Richter, T. Wild, and A. Herkersdorf. Real-Time Capable CAN to AVB Ethernet Gateway Using Frame Aggregation and Scheduling. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2015)*, Grenoble, France, March 2015.

[79] Christian Herber, Andre Richter, Holm Rauchfuss, and Andreas Herkersdorf. Self-virtualized CAN controller for multi-core processors in real-time applications. In *Proceedings of the 26th international conference on Architecture of Computing Systems*, ARCS'13, pages 244–255, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-36423-5. doi: 10.1007/978-3-642-36424-2_21.

[80] Christian Herber, Dominik Reinhardt, Andre Richter, and Andreas Herkersdorf. HW/SW Trade-offs in I/O Virtualization for Controller Area Network. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC) 2015*, San Francisco, June 2015.

[81] Robert Hilbrich. How to safely integrate multiple applications on embedded many-core systems by applying the "correctness by construction"principle. *Adv. Soft. Eng.*, 2012:3:1–3:14, January 2012. ISSN 1687-8655. doi: 10.1155/2012/354274.

[82] Martin Hobelsberger. *Reusability evaluation of component-based embedded automative software systems.* PhD thesis, University of Magdeburg, 2012.

[83] IBM. IBM System/360 Model 67. Systems Reference Library, February 1972.

[84] IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems. Technical Report IEC 61508, The International Electrotechnical Commission, 3, rue de Varembé, Case postale 131, CH-1211 Genève 20, Switzerland, 1998.

[85] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *Software Engineering, IEEE Transactions on*, 31(3):213–225, 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.38.

[86] Intel Corporation. Intel® Virtualization Technology for Connectivity. White Paper 0609/BY/OCG/HOP/750, Intel Corporation, June 2009.

[87] Intel LAN Access Division. Intel VMDq Technology. White paper, Intel Corporation, March 2008. Revision 1.2.

[88] A. Iqbal, N. Sadeque, and R. I. Mutia. An Overview of Microkernel, Hypervisor and Microvisor Virtualization Approaches for Embedded Systems. Technical report, Lund University, 2010.

[89] ISO 26262. Road vehicles - Functional safety - Part 1-10. Technical report, International Organization for Standardization, 2011.

[90] ISO/IEC 25000. ISO/IEC 25000 - Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE. Technical report, ISO/IEC, March 2014.

[91] ISO/IEC 9126. Iso/iec 9126. software engineering – product quality. Technical report, ISO/IEC, 2001.

[92] Dominik Jürgens, Dominik Reinhardt, Rolf Schneider, Georg Hof, Udo Dannebaum, and Andreas Graf. Implementing Mixed Criticality Software Integration On Multicore - A Cost Model And The Lessons Learned. In *SAE 2015 World Congress & Exhibition*, 2015.

[93] R. Kaiser and S. Wagner. The PikeOS Concept - History and Design. Technical report, Sysgo, 2007.

[94] R.S. Kaplan and D.P. Norton. The Balanced Scorecard: Measures that Drive Performance. In *Harvard Business Review*, HBR on point, pages 71–79. Harvard Business School Publishing, 1992. ISBN 9781578514090.

[95] Will Keegan. The Rise of the Type Zero Hypervisor. Technical report, Lynx Software Technologies, Inc., 2012.

[96] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[97] F. Kluge, M. Gerdes, and T. Ungerer. Autosar os on a message-passing multicore processor. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 287–290, June 2012. doi: 10.1109/SIES.2012.6356598.

[98] J.C. Knight. Safety critical systems: challenges and directions. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 547 –550, May 2002.

[99] Winfried Kuehnhauser. Vorlesungsskript: Kommunikationsmodelle. Online, April 2010.

[100] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, NJ, 2006. ISBN 0-691-12202-4.

[101] J.-C. Laprie. Dependable Computing and Fault Tolerance: Concepts and terminology. In *15th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 2–11, 1985. doi: 10.1109/FTCSH.1995.532603.

[102] Jean-Claude Laprie. Dependable Computing: Concepts, Limits, Challenges. In *Proceedings of the Twenty-Fifth International Conference on Fault-tolerant Computing*, FTCS'95, pages 42–54, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7146-7.

[103] M. Masmano, Ismael Ripoll, and A. Crespo. An overview of the XtratuM nanokernel. In *1st Intl. Workshop on Operating Systems Platforms for Embedded Real-Time applications. OSPERT 2005*, 2005. ISBN 121.

[104] Miguel Masmano, Ismael Ripoll, Alfons Crespo, and J.J. Metge. XtratuM: a Hypervisor for Safety Critical Embedded Systems. In *Eleventh Real-Time Linux Workshop*, Dresden (Germany), September 2009.

[105] Giuseppe Mauri. *Integrating Safety Analysis Techniques, Supporting Identification of Common Cause Failures*. PhD thesis, The University of New York - Department of Computer Science, 2000.

[106] J.A. McCall, P.K. Richards, and G.F. Walters. *Factors in Software Quality. Volume I: Concepts and Definitions of Software Quality*. AD A049. General Electric, 1977.

[107] Hans-Ulrich Michel, Dirk Kaule, and Martin Salfer. Vision einer intelligenten Vernetzung. *Elektronik automotive*, 4:28–32, April 2012.

[108] Roberto Mijat and Andy Nightingale. Virtualization is Coming to a Platform Near You. Whitepaper, 2010.

[109] Stanley Milgram. The small world problem. *Psychology Today*, 1(1):61–67, 1967.

[110] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 1965.

[111] B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, 1994.

[112] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *Ninth European Dependable Computing Conference (EDCC)*, pages 132–143, 2012. doi: 10.1109/EDCC.2012.27.

[113] Simon Obermeir. Integration of Virtualized AUTOSAR-Systems on a Multicore-Controller. Master's thesis, University of Applied Sciences Munich, October 2014.

[114] OSEK VDX. OSEK VDX: Open systems and the corresponding interfaces for automotive electronics. ISO 17356-3, November 2005. `http://www.osek-vdx.org`.

[115] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[116] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972. ISSN 0001-0782. doi: 10.1145/361598.361623.

[117] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974. ISSN 0001-0782. doi: 10.1145/361011.361073.

[118] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. Software engineering for automotive systems: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 55–71, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.22.

[119] Holm Rauchfuss, Thomas Wild, and Andreas Herkersdorf. A network interface card architecture for i/o virtualization in embedded systems. In *Proceedings of the 2Nd Conference on I/O Virtualization*, WIOV'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[120] Dominik Reinhardt. Ranking Software Components Using a Modified PageRank Algorithm Including Safety Aspects. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 274–281, July 2014. doi: 10.1109/SAMOS.2014.6893221.

[121] Dominik Reinhardt and Marcus Kucera. Domain Controlled Architecture - A New Approach for Large Scale Software Integrated Automotive Systems. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2013)*, pages 221 – 226, February 2013. doi: 10.5220/0004340702210226.

[122] Dominik Reinhardt and Gary Morgan. An Embedded Hypervisor for Safety-Relevant Automotive E/E-Systems. In *9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 189–198, June 2014. doi: 10.1109/SIES.2014.6871203.

[123] Dominik Reinhardt, Dirk Kaule, and Marcus Kucera. Achieving a Scalable E/E-Architecture using AUTOSAR and Virtualization. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 6(2):489–497, April 2013. doi: 10.4271/2013-01-1399.

[124] Dominik Reinhardt, Daniel Adam, Enno Lübbers, Rakshith Amarnath, Rolf Schneider, Simon Gansel, Stephan Schnitzer, Christian Herber, and Timo Sandmann. Embedded Virtualization Approaches for Ensuring Safety and Security within E/E Automotive Systems. In *Embedded World Conference 2015*, February 2015. ISBN 978-3-645-50144-6.

[125] Dominik Reinhardt, Maximilian Güntner, Marcus Kucera, Thomas Waas, and Winfried Kühnhauser. Mapping CAN-to-Ethernet Communication Channels within Virtualized Embedded Environments. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2015.

[126] Dominik Reinhardt, Maximilian Güntner, and Simon Obermeir. Virtualized Communication Controllers in Safety-Related Automotive Embedded Systems. In *28th International Conference on Architecture of Computing Systems (ARCS)*, March 2015.

[127] I Ripoll, M Masmano, V Brocal, S Peiró, P Balbastre, A Crespo, P Arberet, and JJ Metge. Configuration and Scheduling tools for TSP systems based on XtratuM. *Data Systems In Aerospace (DASIA 2010)*, 2010.

[128] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering*. Viweg + Teubner, Wiesbaden, 4 edition, 2010. ISBN 978-3-8348-0364-1.

[129] Kathrin Scheidemann, Michael Knapp, and Claus Stellwag. Load Balancing in AUTOSAR-Multicore-Systemen Teil 1. *elektroniknet*, 1/2:22–25, January 2010.

[130] Kathrin Scheidemann, Michael Knapp, and Claus Stellwag. Load Balancing in AUTOSAR-Multicore-Systemen Teil 2. *elektroniknet*, 3:21–25, March 2010.

[131] Jörn Schneider and Christian Eltges. Towards an Evaluation Infrastructure for Automotive Multicore Real-Time Operating Systems. In *Proceedings of the 4th international conference on Leveraging applications of formal methods,verification, and validation - Volume Part II*, ISoLA'10, pages 483–486. Springer-Verlag, 2010.

[132] Jörn Schneider, Michael Bohn, and Robert Rößger. Migration of Automotive Real-Time Software to Multicore Systems: First Steps towards an Automated Solution. In *Proceedings Work-In-Progress Session of the 22th Euromicro Conference on Real-Time Systems*, ECRTS'10, pages 37–40, July 2010.

[133] Rolf Schneider, Andre Kohn, Karsten Schmidt, Sven Schoenberg, Udo Dannebaum, Jens Harnisch, and Qian Zhou. Efficient Virtualization for Functional Integration on Modern Microcontrollers in Safety-Relevant Domains. In *SAE 2014 World Congress & Exhibition*, April 2014.

[134] Markus Schöttle. Wir wollen den elektrischen Stromverbrauch halbieren. *ATZ elektronik*, 01/2012:16–19, January 2012. Springer Automotive Media Wiesbaden GmbH (2012).

[135] Wolfgang Schwitzer, Rolf Schneider, Dominik Reinhardt, and Georg Hofstetter. Tackling the Complexity of Timing-relevant Deployment Decisions in Multicore-based Embedded Automotive Software Systems. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 6(2):478–488, April 2013. doi: 10.4271/2013-01-1224.

[136] Freescale Semiconductor. Hardware and Software Assists in Virtualization. Whitepaper, August 2009.

[137] Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 1558609105.

[138] Ian Sommerville. *Software Engineering*. International Computer Science Series. Pearson/Addison-Wesley, Harlow, England, 9 edition, 2011. ISBN 9780137053469.

[139] W. P. Stevens, G. J. Myers, and L.L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974. ISSN 0018-8670. doi: 10.1147/sj.132.0115.

[140] Jon Stokes. *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture*. No Starch Press, San Francisco, CA, USA, 2006. ISBN 1593271042.

[141] Thilo Streichert and Matthias Traub. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug- Modellierung und Bewertung von Echtzeitsytemen*. Springer Vieweg, 2012.

[142] M. Strobl, M. Kucera, A. Foeldi, T. Waas, N. Balbierer, and C. Hilbert. Towards automotive virtualization. In *International Conference on Applied Electronics (AE)*, pages 1–6, September 2013.

[143] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3):202–210, 2005.

[144] Herb Sutter and James Larus. Software and the concurrency revolution. *ACM-Queue*, 09:54–62, September 2005.

[145] A.S. Tanenbaum and H. Bos. *Modern Operating Systems*. Pearson Education, 2014. ISBN 9780133592511.

[146] Jeffrey Travers, Stanley Milgram, Jeffrey Travers, and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[147] Inc. VMware. Understanding Full virtualization, Paravirtualization, and Hardware Assist. White Paper WP-028-PRD-01-01, VMware, Inc., September 2007.

[148] Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmund Ihler, Timo Kehrer, Uwe Mehlig, and Uwe Zdun. *Software-Architektur - Grundlagen, Konzepte, Praxis (2. Aufl.)*. Spektrum Akademischer Verlag, 2009. ISBN 978-3-8274-1933-0.

[149] Barry W. Boehm, Chris Abts, and A. Winsor Brown. *Software Cost Estimation with Cocomo II*. Prentice Hall International, August 2000. ISBN 0130266922.

[150] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. ISBN 9780521387071.

[151] Thomas Wenzel, Martin Fassl, and Joachim Kalmbach. Rezept für sichere Software. *Elektronik automotive*, 11:52–53, November 2010.

[152] M. Zetlmeisl, M. Thomas, D. Hallmann, and E. Merz. BSW partitioning for safety with AUTOSAR R3.2 and R4.0 - a closer look. Technical report, Robert Bosch GmbH, May 2011. 3rd AUTOSAR Open Conference, Frankfurt/Main, Germany.

[153] B. Zimmer, C. Dropmann, and J.U. Hanger. A Systematic Approach for Software Interference Analysis. In *25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 78–87, Nov 2014. doi: 10.1109/ISSRE.2014.12.

Publikationen

Konferenzen

2015 Reinhardt, D.; Güntner, M.; Kucera, M.; Waas, T. und Künhauser, W.; *Mapping CAN-to-Ethernet Communication Channels within Virtualized Embedded Environments*; 10th IEEE International Symposium on Industrial Embedded Systems (SIES); Juni 2015.

2015 Herber, C.; Reinhardt, D.; Richter, A. und Herkersdorf, A.; *HW/SW Tradeoffs in I/O Virtualization for Controller Area Network Results*; 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), Juni 2015.

2015 Jürgens, D.; Reinhardt, D.; Schneider, R.; Hof, G.; Dannebaum, U. und Graf, A.; *Implementing Mixed Criticality Software Integration On Multicore - A Cost Model And The Lessons Learned*; SAE World Congress and Exhibition, April 2015.

2015 Reinhardt, D.; Güntner, M. und Obermeir, S.; *Virtualized Communication Controllers in Safety-Related Automotive Embedded Systems*; 28th International Conference on Architecture of Computing Systems (ARCS); März 2015.

2015 Reinhardt, D.; Adam, D.; Lübbers, E.; Amarnath, R.; Schneider, R.; Gansel, S.; Schnitzer, S.; Herber, C. und Sandmann, T.; *Embedded Virtualization Approaches for Ensuring Safety and Security within E/E Automotive Systems*; Embedded World Conference 2015; Februar 2015.

2014 Reinhardt, D. und Morgan, G.; *An Embedded Hypervisor for Safety-Relevant Automotive E/E-Systems*; 2014 9th IEEE International Symposium on Industrial Embedded Systems (SIES), Juni 2014, 189-19.

2014 Reinhardt, D.; *Ranking Software Components Using a Modified PageRank Algorithm Including Safety Aspects*; 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV); Juli 2014; 274-281.

2013 Reinhardt, D. und Kucera, M.; *Domain Controlled Architecture - A New Approach for Large Scale Software Integrated Automotive Systems*; International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2013), Februar 2013, 221 - 226.

Journalartikel

2013 Reinhardt, D.; Kaule, D. und Kucera, M.; *Achieving a Scalable E/E-Architecture using AUTOSAR and Virtualization*; SAE International Journal of Passenger Cars - Electronic and Electrical Systems, April 2013, Vol. 6, 489-497.

2013 Schwitzer, W.; Schneider, R.; Reinhardt, D. und Hofstetter, G.; *Tackling the Complexity of Timing-relevant Deployment Decisions in Multicore-based Embedded Automotive Software Systems*; SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 2013 April 2013, Vol. 6, 478-488.

Präsentationen

2014 Reinhardt, D.; *Large Scale Software Integration within Domain-Oriented Automotive E/E-Systems*; 1st International Conference on Automotive Embedded Multi-Core Systems; Stuttgart - Deutschland; September 2014.

2013 Farrall, G.; Morgan, G. und Reinhardt, D.; *HW Support for Embedded Automotive Hypervisors*; Keynote; 25th Euromicro Conference on Real-Time Systems (ECRTS 2013), Paris - Frankreich, Juli 2013.

2012 Reinhardt, D.; *Domäne Automotive - Qualifizierung von Systemen mit Multicore-Komponenten*; BICCnet - Bavarian Information and Communication Technology Cluster; München - Deutschland; Oktober 2012.